Self-moving checkers robot

Final Report

M.P. Bonney
u15007172

Submitted as partial fulfilment of the requirements of Project EPR400 in the Department of Electrical, Electronic and Computer Engineering University of Pretoria

November 2019

Study leader: Dr. J. Schoeman

Part 1. Preamble

This report provides a detailed account of the work that I completed in order to design and develop an autonomous checkers playing robot arm that was able to play a game of checkers against a human opponent.

Project proposal and technical documentation
The main report consists of a copy of the approved Project Proposal (Part 2), as well as the technical documentation (Part 4). Part 4 contains the Computer-aided Design(CAD) documentation, circuit schematics, PC board designs as well as the software code. A flash drive containing all the technical documentation is also provided.

Project history
This project does not build on any previous projects

Language editing
This document has been language edited by a knowledgeable person. By submitting this document in its present form, I declare that this is the written material that I wish to be examined on.

My language editor was Mr. L.G. Bonney

Language editor signature *Date*

Declaration

I, _____ understand what plagiarism is and have carefully studied the plagiarism policy of the University. I hereby declare that all the work described in this report is my own, except where explicitly indicated otherwise. Although I may have discussed the design and investigation with my study leader, fellow students or consulted various books, articles or the Internet, the design/investigative work is my own. I have mastered the design and I have made all the required calculations in my lab book (and/or they are reflected in this report) to authenticate this. I am not presenting a complete solution of someone else.

Wherever I have used information from other sources, I have given credit by proper and complete referencing of the source material so that it can be clearly discerned what is my own work and what was quoted from other sources. I acknowledge that failure to comply with the instructions regarding referencing will be regarded as plagiarism. If there is any doubt about the authenticity of my work, I am willing to attend an oral ancillary examination/evaluation about the work.

I certify that the Project Proposal appearing as the Introduction section of the report is a verbatim copy of the approved Project Proposal.

M.P. Bonney Date

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

A	Amperes
ABS	Acylonitrile Butadiene Styrene
AC	Alternating Current
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASIC	Application-specific Integrated Circuit
ARM	Advanced Reduced Instruction Set Computer Machine
ATB	Asus Tinker Board
CAD	Computer-aided Design
CMOS	Complementary Metal Oxide Semiconductor
CNC	Computer Numerical Control
CSI	Camera Serial Interface
CV	Computer Vision
DC	Direct Current
DOF	Degree-of-freedom
GUI	Graphical User Interface
g	Grams
HMI	Human Machine Interface
IDE	Integrated Development Environment
IK	Inverse Kinematics
LED	Light Emitting Diode
Mbps	Megabits per second
MCTS	Monte Carlo tree search
ML	Machine Learning
mm	millimetres
μm	micrometres
μs	microseconds
OpenCV	Open Source Computer Vision library
OS	Operating System
PCB	Printed Circuit Board
PIC	Programmable Intelligent Computer
PID	Proportional-integral-derivative
PLA	Polylactic acid
PWM	Pulse Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
RPM	Revolutions per minute
SBC	Single-board computer
SCARA	Selective Compliance Articulated Robot Arm
TTL	Transistor-Transistor logic
TPU	Tensor Processing Unit
3D	Three- dimensional
2D	Two- dimensional
UML	Unified Modelling Language
Vcc	Voltage at common collector

Part 2. Project identification: Approved Project Proposal

This section contains the problem identification in the form of the complete, approved Project Proposal, unchanged from the final approved version.

For use by the project lecturer	Approved		Revision required	

Feedback

To be completed by the student						
PROJECT PROPOSAL 2019				Project no	JS 13	Revision no 0
Title	Surname	Initials	Student no	Study leader (title, initials, surname)		
Mr	**Bonney**	**MP**	15007172	Dr. J. Schoeman		
Project title	Self-moving checkers robot					

Language editor name	Language editor signature
Lee Bonney	
Student declaration I understand what plagiarism is and that I have to complete my project on my own.	**Study leader declaration** This is a clear and unambiguous description of what is required in this project
Student signature	Study leader signature

1. Project description

What is your project about? What does your system have to do? What is the problem to be solved?

Checkers is a popular two-player strategy board game played on an 8x8 board. The game (also known as Draughts in the UK) can be played against a human opponent or a computer and the main objective of the game is to capture all of the opponent's pieces. Current checkers computers operate using pressure pads, however, these do not provide an interactive gaming experience. Furthermore, these checkers computers require the player move the computer's pieces. This requires an LCD display, LEDs on the board's peripheries as well as a speaker to confirm that the computer's piece has been moved to the correct location. As a result, playing against a traditional checkers computer is arduous and uninspiring as is described in Matuszek et al. (Gambit: An Autonomous Chess-Playing Robotic System, IEEE International Conference on Robotics and Automation, May 2011). The aim of the project is to develop an autonomous checkers robot that is able to move its own pieces and interact with a human opponent in an immersive game of checkers. The system should be able to validate the human player's moves (using a camera), update the current board state after every move, compute a valid move and actuate the move using the robotic arm. The project will be exhibited at the 2020 Tuks open day and will aim to inspire school children to pursue a career in engineering and AI.

2. Technical challenges in this project

Describe the technical challenges that are beyond those encountered up to the end of third year and in other final year modules.

2.1 Primary design challenges

• The design of a computer vision system that uses inverse kinematics in order to determine the real-world coordinates of pieces detected by the camera. This involves inputting the length of the robotic arm joints, the number degrees of freedom of the robot as well as how the joints interact with each other.
• The design of the simulation software that can simulate the movement of the robotic arm in three-dimensional space.
• The design of a game controller and AI that is able to validate the board state as well as compute a valid move.
• The design of a calibration system that can determine the initial board state and calculate if the robotic arm will be able to reach every position on the board.

2.2 Primary implementation challenges

• The implementation of a board state validation system that will be able to identify invalid moves.
• The implementation of a servo controller that will be able to actuate the servos with a high level of precision and control.
• The implementation of a robotic arm that is able to move move smoothly to to the desired coordinates on the checkers board.
• The implementation of a robotic gripper that requires precise control in order to move pieces to new positions as well as remove captured pieces from the board.
• The checkers board size is limited by the reach of the robot arm.
•

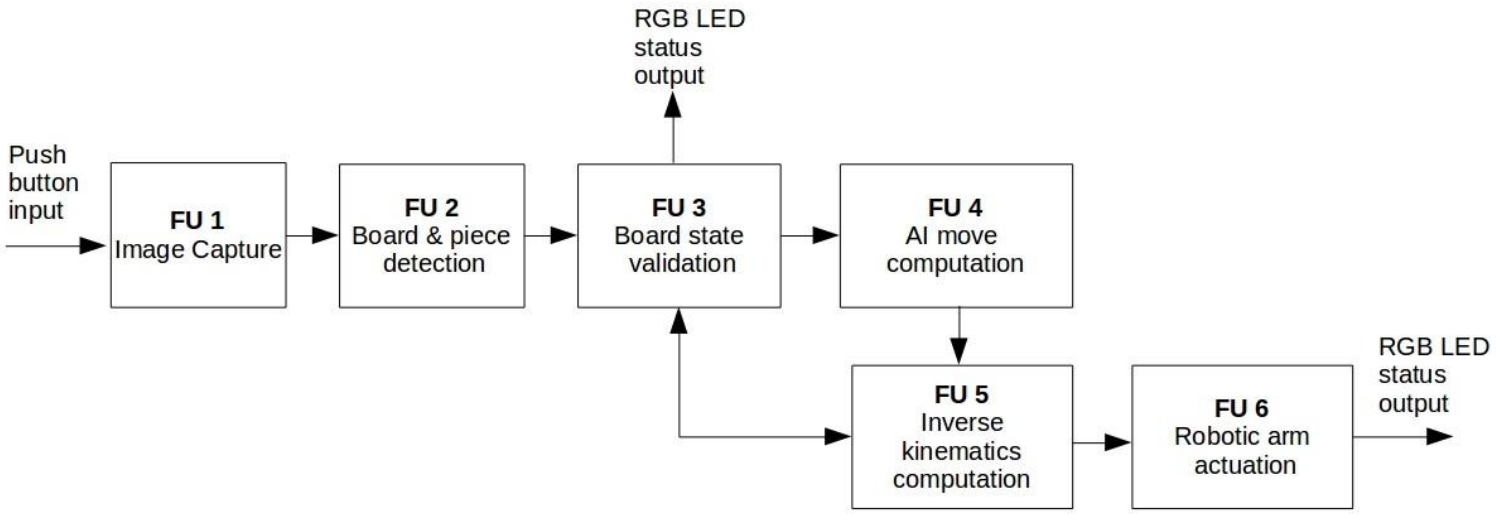
3. Functional analysis

3.1 Functional description

Describe the design in terms of system functions as shown on the functional block diagram in section 3.2. This description should be in narrative format.

Once the human player has made a move, it is confirmed by pressing a push button which activates FU1. FU1, the camera (Placed above the board), captures the board state at the start of the game and at the end of every move. The current board state image is then fed into FU2, the computer vision(CV) system. The CV system performs edge detection to detect the board as well as the squares. The current state is then subtracted from the previous state in order to determine the player move that was made. This new board state is then fed into FU3, the game controller. The game controller, then validates the board state and provides visual feedback to the user about the move validity via RGB LEDs. Should an invalid move be detected, a red LED is activated, the player is required to make a legal move and the system reverts to FU1, once they have made an alternative move and activated the push button again. Should a valid move be detected, a green LED is activated and the board state in the controller is updated and fed to FU4, the checkers engine. The AI then computes a valid move based on the current position. Once the AI move has been computed it is fed into FU5, the inverse kinematics system. The relevant joint parameters necessary to acquire the target coordinates are then calculated using kinematics equations. The target coordinates of a specific move are then validated with the CV system to confirm the validity of the action. Once the actuator parameters have been computed they are fed to FU6, the servo controller. The waist, shoulder, elbow, wrist and gripper servos of the robot arm are then actuated to the desired positions using an appropriate communication protocol. The progress of the robot arm activation is output on the RGB LEDs and notifies the user when it is complete. The RGB LEDs also display the outcome of the game (Player/computer win or draw) at the end of the game.

3.2 Functional block diagram



4. System requirements and specifications

These are the core requirements of the system or product (the mission-critical requirements) summarised in table format .

	Requirement 1: fundamental functional and performance requirement	**Requirement 2**	**Requirement 3**
1. Core mission requirements of the system or product. Solution of the problem will be the most important requirement. Capture this in the set of requirements.	The system is required to move its own pieces to a new position on the checkers board.	The system is required to detect the human player moves as well as capture the current board state.	The system is required to compute a move that is suitable for the given board position.
2. What is the target specification (in measurable terms) to be met in order to achieve this requirement?	Once an AI move has been calculated, it should take no longer than 15 seconds to actuate a single move(ie. Diagonal move with no hops/captures).	The computer vision system should detect the board state and display if it is valid in less than 2 seconds.	The AI should compute a move in no more than 10 seconds.
3. Motivation: how will meeting this specification solve the problem?	The speed at which the autonomous robot arm actuates its moves contributes to the overall gaming experience.	Accurate and efficient detection of the board state and human player moves ensures that the checkers game can be played in real-time, thus enhancing the user experience.	A checkers engine that can rapidly evaluate the position and determine an appropriate move allows a human player to be challenged by the autonomous robot.
4. How will you *demonstrate* at the examination that this requirement has been met?	A stopwatch will be used to measure the time elapsed from commencement of the actuation process to its completion.	A stopwatch will be used to measure the time elapsed from when the push button is pressed to the move validation feedback.	A stopwatch will be used to measure the time elapsed from the commencement of the AI move computation to its completion.
5. What is the deliverable? What are the aspects that you will design and implement yourself to meet this requirement? If none, indicate clearly.	The design and construction of the circuitry and algorithms for the servo controller. The design and construction of the robotic arm and gripper.	The software design and implementation required for the image preprocessing, board detection, square detection and piece detection.	The design and implementation of the algorithm for the checkers engine from first principles. Testing of the AI against other available checkers engines.
6. What are the aspects to be taken off the shelf to meet this requirement? If none, indicate clearly.	Servos with feedback control will be purchased off the shelf.	A camera will be purchased off the shelf. Existing software libraries will be used for the image processing.	No aspect of this design will be taken off the shelf.

System requirements and specifications (continued)

	Requirement 4	Requirement 5	Requirement 6
1. Core mission requirements of the system or product. Solution of the problem will be the most important requirement. Capture this in the set of requirements.			
2. What is the target specification (in measurable terms) to be met in order to achieve this requirement?			
3. Motivation: how will meeting this specification solve the problem?			
4. How will you *demonstrate* at the examination that this requirement has been met?			
5. What is the deliverable? What are the aspects that you will design and implement yourself to meet this requirement? If none, indicate clearly.			
6. What are the aspects to be taken off the shelf to meet this requirement? If none, indicate clearly.			

5. Field conditions

These are the core requirements of the system or product (the mission-critical requirements) summarised in table format .

	Field condition 1	Field condition 2	Field condition 3
Field condition requirement. In which field conditions does the system have to operate? Indicate the one, two or three most important field conditions.	The board may not move during the game.	The pieces are required to be placed directly in the centre of the squares by the human player. The human player may not swap pieces in an attempt to cheat.	The system is required to work indoors with artificial lighting conditions and no external lighting interference.
Field condition specification. What is the specification (in measurable terms) for this field condition?	Board must be perfectly positioned on a marker for the duration of the demonstration. The board needs to be smaller than 20x20 cm in size.	It will be verified that the checkers pieces do not touch the borders of the squares.	No foreign objects/hands may be present on the board when the human player confirms a move.

6. Student tasks

6.1 Design and implementation tasks

List your primary design and implementation tasks in bullet list format (5-10 bullets). These are *not* product requirements, but *your* tasks.

- Development and implementation of a computer vision system.
- Design and implementation of the checkers game controller and HMI.
- Design and implementation of the checkers engine AI.
- Design and 3D printing of the mechanical robotic arm and camera stand.
- Design and implementation of the servo controller PCB.
- Calculation and programming of inverse kinematics equations.
- Simulation of the robotic arm in 3D-space using Python.
- Fine-tuning of PID control system in each servo.
- Integration of hardware and software subsystems on an ARM processor.

6.2 New knowledge to be acquired

Describe what the theoretical foundation to the project is, and which new knowledge you will acquire (beyond that covered in any other undergraduate modules).

- The student will be required to gain an understanding of how to implement a computer vision system to perform image processing.
- The student will be required to study the various inverse kinematics techniques, select the most appropriate method and implement the method in Python in order to map real world coordinates in three-dimensional space.
- The student will be required to acquire an understanding of the simulation software in which the movement of the robotic arm will be simulated.
- The student will be required to evaluate the literature in order to acquire an understanding of developing a checkers AI engine that is capable of providing a challenge to the human opponent.
- The student will be required to gain insight into the mechanical challenges involved in designing and actuating a robot arm, the basics of 3D printing as well as Computer Aided Design (CAD).
- The student will be required to study the control systems theory, specifically PID control parameters, in order to control multiple servos in real time.

Part 3. Main Report

1. Literature Study

Many of the studies that were reviewed, were focused on the game of chess and not checkers, however, the similarities between the two games allows for these studies to be used in the review of literature for the autonomous checkers robot.

1.1 Description of background and context

Luqman et al. [1], described an autonomous four degree-of-freedom(DOF) chess robot that utilised a non-sensory chessboard and unmodified pieces. The system was able to play a chess game autonomously against a human opponent and made use of a web camera mounted directly above the chessboard in order to capture the current game state. The captured coloured images were converted into a greyscale image and stored in a matrix. The computer vision (CV) board detection algorithm operated by performing Canny edge detection on the squares of the chessboard and constructing a segmented frame of the image. Moves were detected by subtracting the current game state image from the previous game state image and calculating the coordinates of the pieces that were moved. Each pixel was analysed and thus the system was highly susceptible to noise, resulting in an increase in overall processing time.

Human player inputs were detected by the CV system using hand motion detection, subsequently, real-time processing of images was required to detect a human player's move. Furthermore, the CV system was not able to operate optimally when people were standing around the board due to the change in contrast of the input images. A major drawback of the CV system was that it was unable to detect illegal moves.

The chess engine and CV system were run on a separate laptop and moves were transmitted to the embedded system via serial communication. The robotic arm eliminated the complexity of the yaw axis motion by placing the base on sliders driven by stepper motors. A direct current (DC) motor was used to control the end effector that manipulated the chess pieces by controlling the motion of the gripper jaws.

Sokic et al. [2], presented a computer vision system for implementation on an autonomous chess robot. A camera was positioned above the chessboard and utilised absolute image subtraction to detect changes in the game state. The brightness of captured frames was equalised to enhance the image contrast and improve the accuracy of the move detection algorithm. The resolution of the captured images was reduced, which yielded an improvement in processing time. The resultant image of the absolute subtraction algorithm was converted into a binary (black and white) image. This was achieved by using a threshold value to differentiate between actual moves and noise.

It was noted that the value of noise in the captured images was similar to the value of an actual chess move. The algorithm had difficulty detecting game states in which multiple squares were involved (for example, when castling). The problem of determining the direction of the chess move was addressed by computing which squares were empty in the latest game state image. Image processing time as well as memory requirements were significantly reduced for move detection by only storing a frame once the human player had made a

move (using a hardware trigger similar to a chess clock). The study noted that unfavourable shadows and low lighting conditions significantly degraded the accuracy of the system and that the best results were achieved using a high-contrast, minimal-gloss playing board.

Matuszek et al. [3], developed a six DOF autonomous chess robot capable of playing against human opponents. The system was able to play with arbitrary chess sets on a variety of different chessboards without requiring any calibration of the pieces. The robot was also able to communicate with the human player in natural spoken language. Real time tracking of the board state was achieved by continuously calibrating the chessboard. A parallel jaw gripper was employed to grasp the chess pieces. Actuators with zero backlash gearboxes and 0.0018-degree precision encoders that used double needle bearings to handle significant cantilever loads were used to construct the shoulder and elbow joints. Three Dynamixel RX-28 servomotors were selected to construct the wrist and a Dynamixel RX-10 in a double crank mechanism was used to construct the gripper. A rubber finger tip was attached to the gripper that allowed it to conform to objects.

A PrimeSense camera that was mounted on torso provided depth information of each pixel of the chessboard image. A web camera taken from an Apple Macbook was mounted onto the gripper to perform chess-piece recognition. A chess piece classifier was trained to identify various chess pieces using a binary support machine vector (SVM). A two-dimensional (2D) image of the chessboard was used to determine the position of the squares. The three-dimensional (3D) points were then fitted to a plane to locate the real-world coordinates, relative to the robotic arm.

A 2D projection of all the pieces on the board was constructed using an isometric projection of all the points above the board. The CV system was also able to detect the colours of the pieces by calibrating the colours of each chess piece before the commencement of a game. The initial configuration of the board was assumed to be correct. A complete move was executed in a mean time of 22.5 seconds. The percentage of successful grasps of perfectly placed pieces made by the autonomous robot was 91.6%. When the pieces were deliberately placed in the worst possible position, the percentage of successful grasps with visual servoing was 77.5% and 17.5% without visual servoing. It was found however, that visual servoing which involved piece detection in real-time, significantly slowed down the game play and was susceptible to changes in lighting conditions. The autonomous robot was developed as a completely embedded system that did not require a display, input devices or any buttons.

Christie et al.[4], engineered a vision system for an autonomous chess playing robot, with all the processing executed on a separate laptop computer. Chessboard detection was achieved by first pre-processing the captured image with Gaussian filtering, followed by applying a threshold function to the image. Hough transformation line detection was used to locate vertical and horizontal lines, the intersection of which were used to identify each square on the chessboard. Lines that were located outside the perimeter of the chessboard were neglected. Image subtraction was used to obtain a difference image and point mapping was able to compute the squares that were involved in the move. It was concluded that normal moves would contain two active cells, three active cells for en passant moves and four active cells for castling. The limitations of this system were the difficulty in detecting a black piece on a

black square, due to the resultant low contrast and the system failing to detect if a piece had been promoted.

Banerjee et al. [5], designed and developed a three DOF robotic manipulator capable of playing chess against an opponent in real time. It was identified that the texture of the chessboard, the colour of squares and pieces as well as the lighting conditions had a major impact on the performance of the image processing algorithms. As a result, red and yellow pieces were used along with a diffused light source. The Shi-Tomasi corner detection algorithm was used to determine the co-ordinates of the corners. The probable corners are computed by dividing the extreme corner coordinates of the board by eight. Canny Edge detection as well as the dilation operation was used to determine if a square was occupied by a piece. The colours of the chess pieces were detected by comparing the piece pixel colours to a threshold value.

Aluminium parts were used to construct the arm due to its low weight and high strength. The robotic arm used the polar coordinate system to actuate the arm to any position on the chessboard. The study observed that the best image processing results were obtained when the chessboard orientation was parallel to the viewing angle of the camera. Furthermore, a degradation in the movement detection was noted when lighting conditions changed between consecutive moves. Grasping of the pieces was achieved using a traditional jaw gripper, which had a tendency to slip when attempting to pick up pieces.

In Chen et al. [6], a humanoid robot was developed to play chess against human opponents. The humanoid consisted of a seven DOF arm which was used to move the pieces. The camera used to perceive the game state was embedded into the arm of the humanoid robot. The arm was actuated to a fixed position whenever the CV system needed to capture the board state. Morphological closing and Gaussian blurring were used in the pre-processing phase, once the board state image had been captured. Otsu thresholding aided in adjusting the threshold based on various lighting conditions. Contours of objects were detected using Canny edge detection in conjunction with dilation to account for discontinuous edges. Squares were extracted from the image by computing the lines that were perpendicular to one another.

The angles of the four chessboard edges were then used to calculate the orientation of the board. This allowed the CV system to process the chessboard image in the event that the board was not precisely parallel to the humanoid robot. Contrast limited adaptive histogram equalisation was used to increase the contrast of the input image so that black pieces could be identified on black squares. A hard-coded threshold was compared to the threshold of each square to ascertain whether the square was occupied by a piece.

The chess engine was able to keep track of the types of pieces by assuming that they start in their initial positions at the commencement of the game. As a result, the type of piece was not required to be detected by the CV system. The humanoid robot made use of a powerful open source chess engine (Stockfish) as well as a chess modelling application (Chessnut), as opposed to developing an artificial intelligence (AI) chess engine and game controller from first principles. An off-the-shelf inverse kinematics (IK) solver (IKFast) was able to calculate all the necessary angles of each joint for a given desired end effector Cartesian coordinate. The robot was capable of dynamically computing coordinates at the

commencement of each move. Subsequently, the chessboard was able to move during the course of a game without compromising the system.

The accuracy of the arm actuation was improved by instructing the arm to move to intermediary points on its way to the final desired coordinate, however, this resulted in a significantly compromised speed of actuation. The response time of the mechatronics system was between forty-five and ninety seconds, depending on the number of pieces involved in the move. The humanoid would occasionally knock over pieces which would require human intervention in order to allow for continuous play. It was concluded that the use of a vacuum gripper could allow the humanoid robot to play other types of board games.

Bailey et al. [7], designed and constructed a checkers robot that could play against a human opponent. It was determined that the CV system was unable to perform optimally in ambient lighting conditions. Thus, a constant light source with a greater intensity than the ambient lighting was mounted at a forty-five degree angle above the checkers board to minimise interference. The conundrum of representing a crowned piece was addressed by using pieces with a king symbol on the bottom side. When a pawn was promoted, the piece was flipped around to reveal a king symbol so that it was recognisable to both the CV system and the human player. Subsequently, the manipulator was not required to actuate two pieces stacked on top of each other.

Due to the low contrast between black pieces and black squares, the pieces needed to be modified. White symbols were painted on black pieces and black symbols were painted on the red pieces. Background subtraction was used to detect moves and identify the different shapes of the symbols on each piece. User input was required to calibrate the system by selecting the top left and bottom right corners of the board, thereafter, the checkers engine provided a random legal move for a given position.

A Cartesian coordinate arm (similar to a CNC machine) that did not require complex trajectory planning was selected over a more complex anthropomorphic arm design. It was calculated that the minimum gripper accuracy needed to be three millimetres. A toothed belt attached to stepper motors was used to drive the horizontal axes coupled with a worm drive to control the vertical axis. An electromagnet was used to pick up the pieces, however, this required that the pieces be constructed with a ferrous material. The stepper motors were controlled using an open loop system with the use of a step waveform generator produced by a microcontroller. The stepper motors would occasionally skip steps, consequently, pieces would offset from the centre of the squares and compromise the entire system.

Danner et al. [8], produced a visual chess recognition system that was able to identify a chessboard as well as the chess pieces. The camera was mounted at a forty-five degree angle to the chessboard, which enabled the shape of different chess pieces to be recognised. A chessboard containing red and green coloured squares was used to aid in detecting light pieces on light squares and similarly, dark pieces on dark squares. The chessboard recognition algorithm involved detection of edges by thesholding the input image, provided that the image did not consist of any red or green objects.

Morphological edge detection was performed on the resultant image, which yielded

a difference image. The Hough transformation algorithm was then applied to the difference image to locate the eighteen lines of the chessboard. Squares were inferred from the intersecting lines that were detected. Chess pieces were identified using the Fourier descriptors which obtained the contours of each shape and compared the shape to a database in the domain of the descriptor. It was found that glare on the pieces due to variable lighting conditions compromised the accuracy of the recognition algorithm.

Unger [9], designed and developed a four DOF autonomous chess-playing robot arm. Line detection with image segmentation was used to detect the outline and squares of the chessboard. Moves were detected using colour histogram equalisation that generated a histogram for every square. The resulting histograms were compared by computing the difference between the probability distributions of each histogram. Although the method provided a high degree of accuracy, a significant processing time of approximately 3.5 seconds was observed for each move. Manual measurements of the chessboard dimensions were used for the IK calculations as the CV system was unable to obtain the exact real-world coordinates required to locate the squares.

A pincer mechanism was used to manipulate the chess pieces. An error of between two and eight millimetres was observed in the vertical axis when testing the precision of the robotic arm. The error increased proportionally with the distance from the base of the robot arm. Since the servomotors did not provide any feedback to the main controller, the movement of the robotic arm was consistently jerky and uncontrolled. It was noted that the servomotors did not operate to the specifications provided by the manufacturer. The system was unable to perform pawn promotions due to the fact that it required information about the removed pieces. The robotic arm for the project was originally intended to be purchased, however, it was discovered that the available off-the-shelf robotic arms were either too expensive or did not meet the project specifications. Ultimately, it was concluded that a superior mechanical arm with feedback would have resulted in a solution that met the requirements.

Brown et al. [10], proposed a gripper (known as the universal gripper) that used a single mass of granular material that could conform to the shape of the object being manipulated. Objects were able to be gripped by applying a vacuum to the granular material, resulting in the compacting and hardening of the gripper. This simple method was able to efficiently and effectively grasp objects without requiring feedback. The approach reduced the number of degrees of freedom (DOF) of the robotic arm, thus reducing cost, complexity as well as the effective gripping speed.

The suction effect as well as the interlocking of the object and the gripper resulted in a holding force that significantly exceeded the weight of the manipulated object. Furthermore, only a portion of the object's surface was required to be gripped in order to hold it securely. Prior information about the objects that were being manipulated was not required. A gripper bag containing coffee grounds as the granular material was found to be the most effective when compared to other granular materials such as sand. The coffee grounds had a low density which meant that the gripper membrane was not strained by the weight of the granular material.

1.2 Summary of application of background literature

A major flaw in both [1] and [4] was that the image processing and move computation were executed on a separate laptop as opposed to a single embedded system. This issue was addressed by employing the use of a single board computer (SBC) as well as a microcontroller that were able to perform all the processing required by the system.

Canny edge detection was successfully implemented in [1], [5] and [6] to locate the squares of the board. The Hough transformation was equally as effective in detecting lines in [4] and [8]. Conversely, the board detection algorithms utilised in [3], [8] and [9] required higher quality input images, which increased the computational complexity, processing time and cost. A combination of both Canny edge detection and the Hough transformation were used in the implementation of the CV system for the checkers robot.

The piece recognition algorithms developed in [3], [7] and [8] increased the overall processing time of the CV system. Furthermore, they were highly susceptible to noise and varying lighting conditions. A significant improvement in processing time and accuracy was achieved in [1], [2] and [4] by placing the camera directly above the board, and using absolute image subtraction for the movement detection. This was the method implemented in the autonomous checkers robot since the initial conditions were known and thus the game state could be tracked by the checkers engine.

It was found that the absolute image subtraction techniques employed in both [7] and [9] attempted to use open-loop control of the actuation system. This was proven to be inaccurate and inconsistent, due to real-world imperfections present in the system such as backlash and frictional forces. As a result, servomotors with feedback were used to actuate the robotic arm. The approach of using hand motion detection in [1] and [3] to detect if a move has been made was found to be computationally expensive and unnecessary. The approach of a hardware trigger used in [2] and [7] was found to be more efficient. Therefore, a simple push button was used to confirm the completion of a move made by the human player.

In [4], [5], [7] and [8] it was observed that the detection of black pieces on black squares and white pieces on white squares was problematic for the CV system to process. This low contrast issue was solved by ensuring that the colours of the pieces and the squares were different. The histogram equalisation technique implemented in [6] and [9] presented an alternative solution, however, at the cost of significantly increased processing time. As a result, the approach used in [4], [5], [7] and [8] was implemented in the CV system of the autonomous checkers robot.

The game engines utilised in [3], [6] and [9] were taken off-the-shelf, due to time constraints. The autonomous checkers robot improved on this by developing the checkers game controller as well as the AI engine from first principles.

The complexity of an anthropomorphic arm was eliminated in both [1], [5] and [7]

by limiting the movement of the arm to the Cartesian coordinate system. Although this simplifies the problem, it detracts from the immersive game-playing experience. Thus, a four DOF, anthropomorphic and non-intrusive robotic arm was utilised in the autonomous checkers robot.

The gripper jaw mechanisms implemented in [1], [3] and [9] did not provide the consistent manipulation of the pieces due the fact that they required a repeatability equivalent to that of an industrial grade robotic arm. The approach presented in [10], was able to effectively solve the problem by using a universal gripper. This simple, low-cost solution was utilised in the production of the autonomous checkers robot.

The robotic arms developed in [5], [6], [7] and [9] required an inordinately long time to execute moves. Furthermore, the actuation of the arms was often imprecise and uncontrolled. The robotic arm developed in [3] produced satisfactory results, however, it required a substantial budget ($18 000 in parts) and was funded by Intel. The autonomous checkers robot improved on these studies by implementing a low-cost robotic arm that was actuated in a smooth, controlled fashion and provided an immersive gaming experience.

2. Approach

The implementation of the self-moving checkers robot required the successful integration of the CV system, checkers engine and the robotic arm. The capturing of the input images was accomplished using a *Raspberry Picamera* mounted directly above the checkers board. The *Picamera* was selected for its compact size, which allowed it to be mounted onto a gantry frame. The camera was mounted directly above the board to reduce the effects of noise in the input images. The input images were cropped to the size of the checkers board and greyscaled, since the colour of the pieces/board was not required. The checkers board was calibrated by locating and storing the inner corners of the board using the *OpenCV findChessboardcorners* calibration algorithm.

The orientation of the calibrated coordinates was validated by confirming that the first coordinate detected was in the upper section of the captured image. Absolute image subtraction was used to obtain a difference image containing the circles that were involved in a given move. The *OpenCV HoughCircles* algorithm was used to determine the Cartesian coordinates of the circles in the difference image. A virtual grid was constructed using the coordinates of the first corner detected in the calibration algorithm. The coordinates of the circles detected in the difference image were then mapped to a specific row and column of the virtual checkers board. The predicted squares were then passed to the game controller which was able to compute the move that was made as well as whether the move was legal. The game state in the game controller was then updated to reflect the game state of the physical board.

The AI checkers engine was implemented using the *Alpha-Beta pruning* algorithm. The algorithm expanded all of the possible game states of the current position to a depth of three plies ahead. This was found to be optimal in providing a suitable playing strength as well as a fast processing time. A heuristic function evaluated the value of a given game position and selected the optimal move by locating the position was the highest value. Machine learning (ML) approaches were also considered, however, it was found that these did not perform to the level of the *Alpha-Beta pruning* algorithm given the time and resources available.

The robotic arm needed to be capable of executing moves generated by the AI on a physical checkers board. The robotic arms available off-the-shelf were either too expensive or did not meet the requirements for the project, thus the robotic arm needed to be designed and developed from first principles. Initially, the design of a Selective Compliance Articulated Robot Arm (SCARA) that moves in the x-y plane (using polar coordinates) and uses a linear actuator for movement in the z axis was considered, however, the arm did not have the freedom of movement required to move to every position on the checkers board. Furthermore, the z axis movement could only be executed once the correct x and y coordinates had been acquired, resulting in increase in overall actuation time. Subsequently, an articulated robot arm design, capable of moving to any point (within reach) in 3D space was selected.

Four DOF was the minimum number of joints required to actuate the articulated

arm to a desired x, y and z coordinate and thus a four-axis design was selected. The maximum reach of the robot arm was limited to 300mm, which meant that the checkers board was required to be smaller than an area of $200mm^2$. The reach needed to be limited due to the limitations of the actuators that could be acquired with a low budget. The robot arm was originally going to be 3D-printed, however, it was decided to rather use lightweight aluminium tubing which provided a more rigid arm that was required for high precision actuation. The selection of the actuators involved calculating the torque requirements for a four DOF arm design. It was discovered that motors with a high resolution and high torque were required. Additionally, the motors used in the elbow and wrist joints needed to be lightweight to reduce the forces acting on the shoulder joint.

Stepper motors available off-the-shelf did not meet the resolution and weight requirements. Furthermore, the torque supplied by a stepper motor reduces as the speed of the motor increases. Conversely, a servomotor is able to supply the same amount of torque (within its limits), regardless of the speed of operation. Subsequently, it was decided to use servomotors that were able to meet the weight, torque, speed and resolution requirements. Servomotors that provided feedback control were required to account for the high moments of inertia inherent in the cantilever design of an articulated robot arm. Joints that weren't subjected to high forces such as the wrist actuator did not require feedback and open-loop control could be used. Therefore, a traditional hobby servomotor was used for the wrist joint and the *Dynamixel XL430-W250-T* servomotors were selected for the waist, shoulder and elbow joints. Initially, direct drive on all of the actuators was used, however, a belt reduction was later added to the waist axis that doubled its resolution. Two servomotors were used to drive the shoulder joint to double the effective torque generated by the shoulder axis.

The IK calculations computed the matrix of joint angles required to articulate the arm to a specific Cartesian coordinate. The waist axis angle as well as the calculated effective reach required could be calculated using trigonometric equations. The shoulder, elbow and wrist angles were calculated by working in the z plane and computing the angles based on the known lengths of the limbs along with the effective reach of the robot arm.

A gripper needed to be developed to manipulate the pieces and execute the AI move. Initially, an electromagnetic gripper was considered due to its fast pick up and placement speed as well as its ability to compensate for possible errors inherent in the articulation of the robot arm. The problem encountered, was that ferrous pieces would be required and manipulating two stacked pieces would not have been possible. Traditional claw and pincer designs containing foam/rubber tips were also considered. The major drawback of these designs was that they required a repeatability and accuracy that was unattainable with the available resources. Furthermore, additional servomotors would be necessary to control the claw mechanism, adding extra weight to the end-effector. Therefore, a suction gripper containing a bag filled with ground coffee was used to manipulate the pieces in a simple, fast and effective manner. A servomotor was used to control the suction by adjusting the position of a syringe.

The processing of the CV data, checkers game controller, AI engine and IK equations took place on the *Asus Tinker Board S* (ATB). The ATB was selected as it contained a

32 bit Advanced Reduced Instruction Set Computer Machine (ARM) architecture suitable for embedded applications. A lightweight *Linux Debian 9* operating system (OS) was run on the processor which allowed *Python 3.5* scripts to be executed. Real-time performance was required to output clean pulse width modulated (PWM) signals to the servomotors. The fact that *Python* periodically needed to perform memory management tasks, coupled with background kernel scheduling decisions conducted by Linux resulted in compromised real-time performance. As a result, the ATB was interfaced with a Programmable Intelligent Computer (PIC) microcontroller that provided the necessary real-time performance.

3. Design and implementation

This section provides a detailed description of the design and implementation of the subsystems specified in the functional system block diagram in section 3 of the project proposal.

3.1 Image Capture

3.1.1 Theoretical analysis

The acquisition of quality input images was crucial in developing a CV system capable of tracking the game of checkers. The first step was to select a camera suitable for the application of detecting changes in the game state of checkers board. Since only the shapes of the checkers board squares and pieces needed to be detected by the CV system, a camera with a resolution of 640x480 or lower would suffice. The focal length of the camera determines the distance that the camera needs to be from the board, it is measured in mm from the point of convergence in the lens to the camera sensor. Finally, a Complementary Metal Oxide Semiconductor (CMOS) camera sensor was desired as it is able to process images with a high efficiency and low power consumption. The *Raspberry Pi* NoIR camera was selected for its compact size, CMOS image sensor, small focal length and high image transfer rate. The night vision capabilities of the camera were not exploited as they were not required for the application.

Using the camera's focal length, the number of pixels in the image and the size of the image sensor, the minimum mounting height of the camera can be calculated. Since the camera focal length is known to be 3.04 mm, the real world width of the checkers board was 192 mm. The size of the object on the image sensor can be computed as the number of pixels of the object is known. The pixel size of a full resolution (3280x2464) image was 1.12 μm^2. The full resolution of the camera was not required and the lowest resolution of 640x480 was used for the video feed. The optical width of checkers board, scaling factor and focal ratio were 380 pixels, 3280/640 and 2 respectively. Using these values, the size of the object on the image sensor could be calculated as shown in equation 1.

$$\begin{aligned} i &= \frac{p * n * s}{r} \\ i &= \frac{0.00112 \cdot 380 \cdot 5.125}{2} \\ i &= 1.0906\, mm \end{aligned} \tag{1}$$

i: Size of the object on the image sensor
p: Pixel width
n: Number of pixels
s: Scaling factor
r: Focal ratio

Using the size of the object on the image sensor computed in equation 1, the pixels

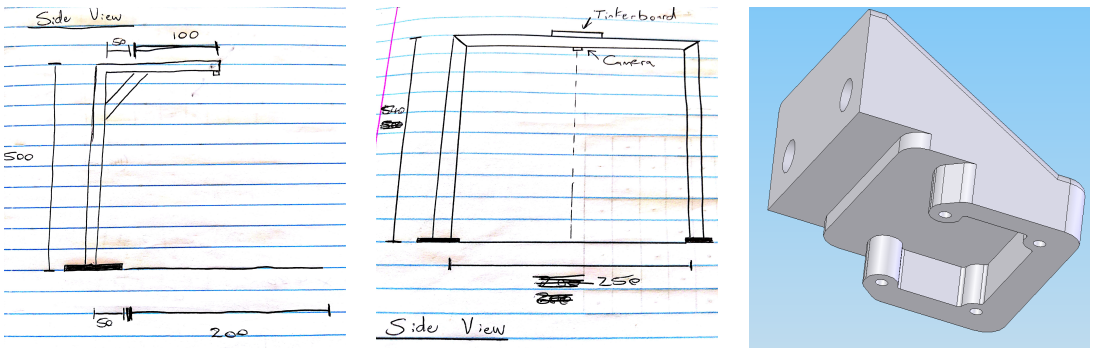
per mm could be calculated using equation 2.

$$d = \frac{r * f}{i}$$
$$d = \frac{192 \cdot 3.04}{1.0906} \tag{2}$$
$$d = 535.19\,mm$$

d: Real-world distance to the object
f: Focal length of the *Raspberry Pi* NoIR V2 camera
r: Real-world width of the object
i: Size of the object on the image sensor

3.1.2 Hardware Design

When the above calculations were applied in the real-world, it was found that a minimum height of 540 mm was required to capture the full board. By adjusting the focus of the camera, the minimum mounting height was ultimately reduced to 480 mm above the checkers board. Mounting the camera onto the robotic arm was ruled out, as the effective height of the robot arm when fully extended was 375 mm, therefore a camera stand was required to mount the camera. The design of the stand needed to be both rigid and stable. Due to the height of the stand required, 3D printing the stand was not a viable solution, subsequently, it was decided to construct the stand using aluminium t-slot extrusions. The first iteration of the mounting stand design shown in figure 1a, was found to be unstable as it only supplied support from a single base. The design in figure 1b supported the frame with two bases, providing superior stability. The housing design shown in figure 1c allowed the camera to be securely attached to the top of the stand. Polylactic acid (PLA) 3D printing material was selected in favour of Acylonitrile Butadiene Styrene (ABS) material. PLA is less susceptible to warping while printing as a lower bed and nozzle temperature can be used. Furthermore, PLA is able to be printed using a low cost 3D printer that does not require an enclosure and ventilation.



(a)
First iteration of the camera stand design

(b)
Second iteration of the camera stand design

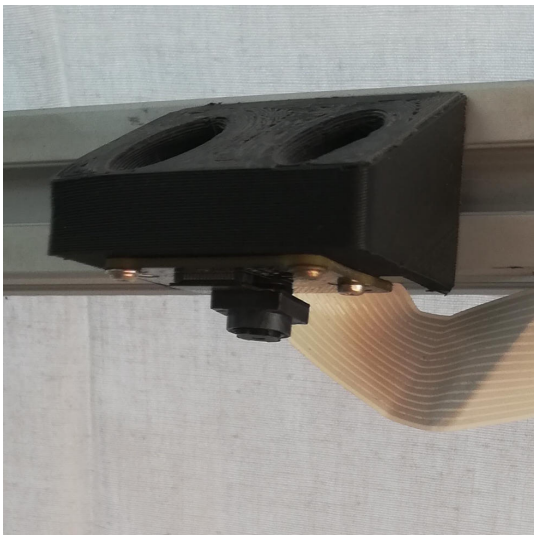
(c)
***SolidWorks* design of the camera housing design**

Figure 1.
Hardware design of the camera stand

3.1.3 Hardware Implementation

The final camera stand and housing implementations are shown in figures 2a and 2b respectively.

(a)
Final implementation of the camera stand



(b)
Final implementation of the camera housing

Figure 2.
Final hardware implementation of the image capture subsystem

3.1.4 Software Design

The *Raspberry Pi* NoIR Camera was able to interface with the ATB using a Camera Serial Interface (CSI). A pipeline based multimedia framework called *GStreamer* was used to capture a live 640x480 video stream that could be accessed in *Python 3.5*. Since the game state could only change at the end of every move, it was unnecessary to process frames from the live stream between moves. Therefore, frames were only captured and processed at the end of every move. A hardware trigger was used to detect the completion of a user move, while the completion of the robot's move was processed following the robot arm's actuation procedure. The block diagram in figure 3 illustrates the procedure taken to capture and pre-process input images. The camera was first initialised by setting up the camera input to a 640x480 video stream. The video stream contained colour information that was not required for the application. Subsequently, once an input image was captured it was greyscaled using the *cvtColor* algorithm in the *OpenCV* library to obtain a monochrome image.

The image was then cropped to the size of the checkers board to reduce the number of pixels that needed to be processed. The calibration procedure used the *findChessboardCorners* to detect the inner corners of the squares on the checkers board. The *findChessboardCorners* algorithm was preferred over the *Harris Corner detection* algorithm as it discarded the corners detected outside the bounds of the checkers board. The orientation of the board was confirmed by computing whether the coordinates of the first corner detected

were smaller than the midpoint of the captured frame. Should the calibration be successful, the coordinates of the corners were stored, otherwise the calibration algorithm was recursively called until the board was detected. The coordinates of the corners were stored in a text file to eliminate the need to calibrate the board prior to the commencement of every game. The input detection algorithm stored the most recent captured frame from the video stream when the state of the hardware trigger was low. Figure 3 shows the block diagram of the flow of data through the image capture subsystem.

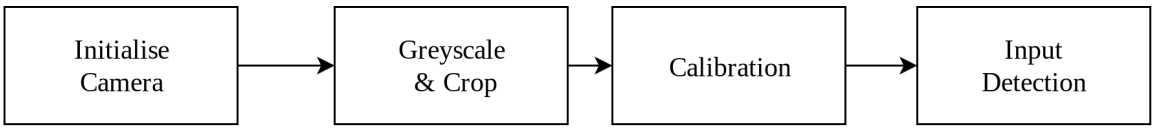


Figure 3.
Block diagram of the image capture subsystem

3.1.5 Software Implementation

All software for the image capture subsystem was developed in *Python 3.5* and executed on the ATB. When performing the calibration procedure, all of the pieces needed to be removed from the checkers board. This ensured that the edges of the pieces did not interfere with the detection of the corners of the squares.

3.2 Board and piece detection

3.2.1 Software Design

The challenge of correctly detecting the board state can be broken down into two stages: The first stage is detecting the squares on the board and the second stage is detecting which squares were occupied by a piece. Many approaches attempt to detect the edges of the board and calculate the intersection of the edges to ascertain the positions of the squares on the board. A simpler, more effective approach that did not require edge detection was taken when detecting the location of the squares on the checkers board. The stored corner coordinates obtained in the calibration procedure were used to mathematically construct a virtual grid of the checkers board. The first step was to calculate the height and width of the squares, since the aspect ratio of the camera was 4:3, the squares actually appeared as rectangles to the camera. The equations for the square width and height are shown in equation 3.

$$\begin{aligned} Square\ width &= 2nd\ coordinate\ x-value - 1st\ coordinate\ x-value \\ Square\ height &= 8th\ coordinate\ y-value - 1st\ coordinate\ y-value \end{aligned} \tag{3}$$

The square width is simply the difference between the first and second coordinate's x values. The square height is computed as the difference between the eighth (single row down from the first coordinate) and first coordinate's y values. The distance of the board from the edge of the frame is calculated by subtracting the square width and square height from the x and y values of the first coordinate to obtain the board offset. The problem of the checkers board not being horizontally aligned can be solved in multiple ways. One way is to rotate the input

image by the angle of the angle between the horizontal line and the checkers square lines, however, this would need to be executed on every input image and significantly increase the overall processing time. An alternative method is to calculate the angle using trigonometry and essentially rotate the virtual grid by the computed angle rather than the actual image. This was the approach that was taken and was achieved using equations 4 and 5.

$$\begin{aligned} Triangle\ width &= 7th\ coordinate\ x-value - 1st\ coordinate\ x-value \\ Triangle\ height &= 7th\ coordinate\ y-value - 1st\ coordinate\ y-value \end{aligned} \tag{4}$$

$$\theta = arccosine\left(\frac{Triangle\ height}{Triangle\ width}\right) \tag{5}$$

The width of the triangle calculated in equation 4 is the effective distance between the first and last column of the checkers board. The triangle height is the parameter that should be zero if the first coordinate (first column) was parallel to the seventh coordinate (last column). The rotational angle is calculated in equation 5 using the inverse cosine of the triangle height and width.

The second stage of the process was to detect the squares containing pieces. One approach is to determine the location of all of the pieces after every move, however, this is unnecessary, as only the changes in the board state need to be tracked. Furthermore, the colour and status (crowned/uncrowned) of the pieces did not need to be detected by the CV system, since the movement of the pieces could be tracked by assuming the correct initial configuration of the game. Absolute image subtraction of the current and previous game states was used to obtain a difference image. An illustration of the principle taken from [4] is shown in figure 4.

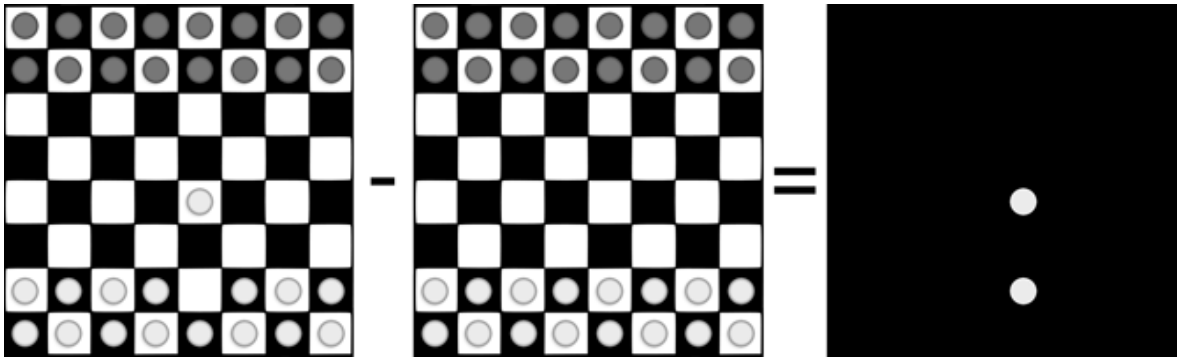


Figure 4.
Illustration of the absolute image subtraction algorithm taken from [4]

The principle of the absolute image subtraction algorithm is that only the changes to the game state need to be tracked. The *OpenCV absdiff* algorithm was used to obtain the resultant difference image. The challenge of deriving the coordinates of the pieces that were moved in the difference image can be solved using many different approaches. One approach is to search through every pixel in the image and use a binary threshold to determine if the pixel is white or black. A prediction of the most probable coordinates of the pieces moved can be made by locating regions of the binary image that contain the most white pixels. The squares

in which the clusters of white pixels are found can then be computed by segmenting the image into a virtual grid and summing the number of pixels found in each square. The squares containing the most white pixels could then be predicted as having been active in a move.

Many problems were encountered when attempting to implement this method in practice. The first issue was that every image contained noise that in some cases could not be distinguished from relevant, useful information. Consequently, false predictions of squares that were inactive in a move were made and squares that were involved in a move went undetected. Furthermore, significant processing time was required to search through and classify every pixel in the image. Thus, an alternative approach was required to effectively solve the problem. Since it was known that the difference image of the pieces involved in a move would always yield a circular shape, this property was exploited to predict the squares involved in a move.

The *OpenCV GaussianBlur* algorithm was used to blur the image, since only the shapes of objects needed to be detected. The *OpenCV HoughCircles* algorithm was then used to obtain a prediction of all of the circles present in the image. The parameters in this algorithm needed to be fine-tuned to ensure that the desired circles were detected. The algorithm first determines the edges present in the image and then computes the gradient of the detected lines. The calculated gradient is compared to the the equation of a circle to obtain a prediction on the circles present in the image. The various parameters, their values and an explanation of the parameter is provided in table 1.

Parameter	**Value**	**Explanation**
dp	1	The inverse ratio of the accumulator resolution to the image resolution.
minDist	20	The minimum distance between the centres of the detected circles.
param1	30	The higher threshold passed to the Hysteresis Thresholding algorithm.
param2	15	The accumulator threshold for the circle centres at the detection stage.
minRadius	10	The minimum radius of the detected circle.
maxRadius	20	The maximum radius of the detected circle.

Table 1.
Parameter values selected for the *HoughCircles* algorithm

A value of one was selected for the inverse ratio of the accumulator resolution to image resolution as the accumulator resolution needed to be equivalent to the image resolution. A minimum distance of twenty pixels between the radii of detected circles was selected as the average circle radius was approximately fifteen pixels. The calculated square width and heights averaged pixel values of forty-five and fifty respectively. Thus, to ensure that detected circles did not overlap, a minimum distance of approximately half the square height and width was selected. The first parameter of the hysterisis thresholding algorithm determines the maximum threshold for detected edges. If the value is too high, too many unwanted edges would be detected. Thus, it was found through experimentation that a value of thirty was

appropriate. The second parameter is required to be half the value of the first parameter to set the lower threshold of the algorithm, thus a value of fifteen was required. Since the average value of the radius of detected circles was fifteen pixels, the minimum and maximum radii of the circles was set to ten and twenty pixels respectively.

The *HoughCircles* algorithm outputs a vector of the detected circles. Each vector contains three elements, the x and y coordinates of the detected circle as well as the radius of the detected circle. Each vector is appended to a list of containing all of the circles detected. The first step was to determine the number of circles detected by the algorithm. If the incorrect number of circles was detected, the move would be flagged as invalid and the user would be required to input their move again. In the case of the detection of a capture move, the three circles are sorted in the order of their x coordinate pixel values. This was useful in determining the middle circle, which was deduced to be the captured piece.

Once the coordinates of the all of the detected circles had been obtained, a prediction of the squares that were active in the move can be ascertained. The row and the column of the predicted squares is calculated in equation 6.

$$\begin{aligned} column &= \frac{(Circle\ coordinate\ x-value)-(offset\ width)}{square\ width} \\ row &= \frac{(Circle\ coordinate\ y-value)-(offset\ height)}{square\ height} \end{aligned} \tag{6}$$

The column of the predicted square is calculated by subtracting the offset width from the x coordinate of the predicted circle and dividing the result by the square width. Similarly, the row of the predicted square is calculated by subtracting the offset height from the y coordinate of the predicted circle and dividing the result by the square height. This yields a float (irrational number) value that is rounded down to obtain the integer row and column predictions. The process is then repeated for all of the detected circles. The row and column of all of the squares involved in the move are then passed to the game controller. The order of the move is calculated by iterating through all of the squares and determining which of the squares currently contain a piece. The squares containing pieces are either captured pieces or the piece being moved. Therefore, the target square can be deduced as the square that does not presently contain a piece. Figure 5 illustrates the interaction of the subsystems that were designed.

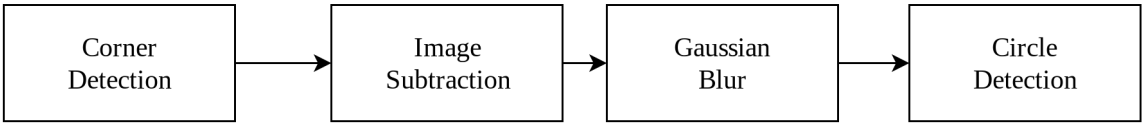


Figure 5.
Block diagram of the board and piece detection subsystem

3.2.2 Software Implementation

All of the software for the board and piece detection subsystem was developed in *Python 3.5*. The algorithm developed was able to successfully detect the board state as well

as changes in the board state. There were conditions in which the CV system was unable to operate optimally and these will be discussed over the next few pages. The move is barely noticeable to the human eye due to the extremely low contrast of the black pieces placed on the black squares. Subsequently, the move made was unable to be detected by the CV system. The problem can be solved by changing the colour of the checkers board to a colour that contrasts with black (For example red). Alternatively, lighter pieces can be used to increase the contrast, this was the approach that was taken.

Figure 6 shows the unified modelling language (UML) class diagram of the *CaptureBoard* class which incorporates both the image capture and board detection subsystems. Variables and functions were made public so that they could interact with other classes when necessary. Figure 7 depicts a flow diagram of the CV system, Once the corners are detected and the correct orientation is verified, the coordinates are stored. Following the processing of the image, the algorithm determines whether circles were detected. A red LED is illuminated if no circles were detected, otherwise, the circles that were detected are sorted and a prediction of the user's move is made. The move is then validated by the board state validation subsystem.

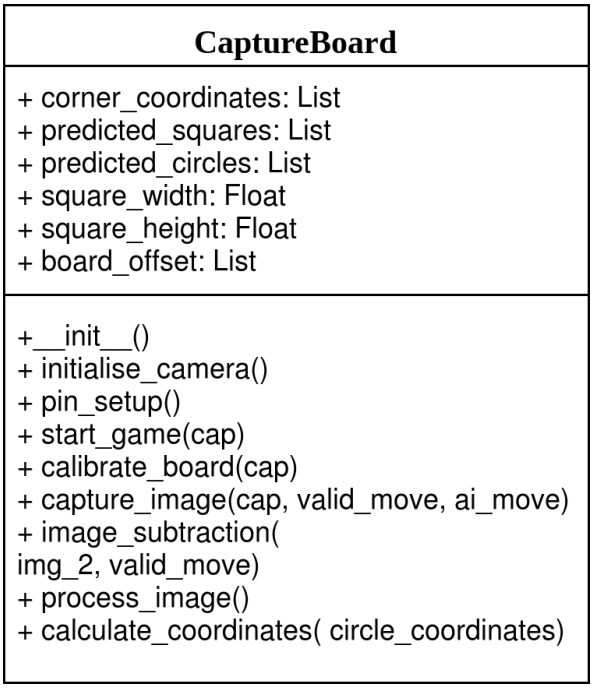


Figure 6.
UML class diagram of the *CaptureBoard* class

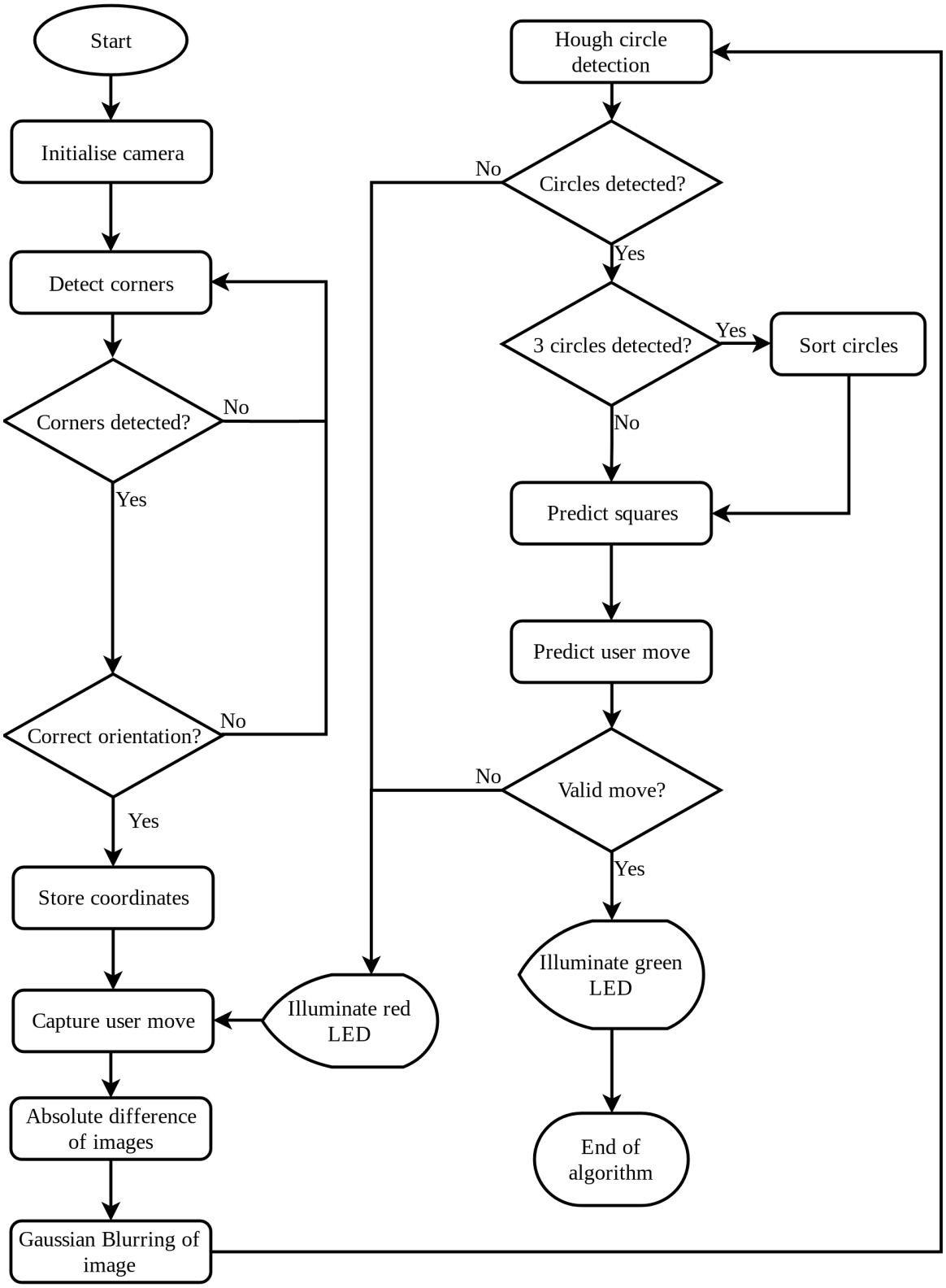


Figure 7.
Flow diagram of CV algorithm

3.3 Board state validation

3.3.1 Software Design

The design of the board state validation subsystem involved developing a checkers game controller that was capable of validating moves as well as updating the game state. A virtual representation of the board and pieces must be developed, along with the logic required to understand the manner in which the pieces can move. There are many approaches that can be used to solve the problem. One approach, is to represent the board as a 2D array of integers, with each integer representing a different state of a specific square. An uncrowned, black piece would be represented by the number '2' and an empty square represented by a '0' for example. When the state of the board changes, every integer in the 2D array needs to be updated. The problem with this simplistic approach is that the integer value of each square needs to be repeatably checked in order to update the board state. As a result, the algorithm is highly inefficient as the entire board needs to be updated after every move.

A more elegant and efficient solution is to use an object-oriented programming approach. Rather than representing each square as an integer value, a 2D array of square objects is created that contain attributes such as colour, row, column, number and coordinates. The piece attribute of each square is a piece object that contains the attributes of the piece such as the colour and the status (crowned/uncrowned) of the piece. To construct the virtual checkers board, every second square in the matrix is initialised to the colour black and populated with the appropriate piece. This approach allows for the seamless and efficient rearrangement of pieces by simply changing the value of the attributes of the desired square. Figure 8 illustrates the final design approach that was taken to detect the board state.

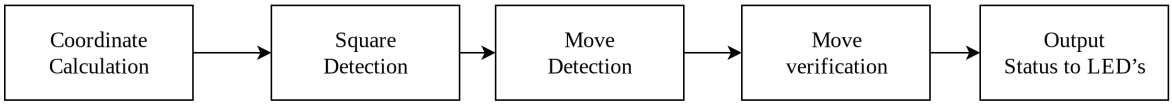


Figure 8.
Block diagram of the board state validation subsystem

3.3.2 Software Implementation

Figure 9 depicts the flow diagram for the *Board* class. The board is initialised by populating the first three rows with black pieces and last three rows with white pieces. The algorithm first checks whether the origin square contains the correct piece as well as if the desired target square is vacant. Should these two conditions not be met, the move sequence is rejected and the move is invalid. Once the origin square is known, all of the possible moves for that specific piece are computed. The status and colour of the piece are then used to compute all of the possible single and capture moves for the given position. Each move in the list of possible moves is compared to the move that was made by the user to confirm that the move was valid.

Once the move has been validated, the board state can be updated. This is achieved by moving the piece that was on the origin square to the target square. Should a target square be in either the first row (for white) or the last row (for black), the piece located on the target square is crowned. Captured pieces are removed from the board by removing the piece attribute of the captured square. Figure 10 shows the UML diagram of the board validation subsystem. The *Board* class is the parent of both the *Piece* and *Square* classes meaning that they inherit all the properties, behaviours and attributes of the parent class. More specifically, a composite aggregation (binary association) exists between the parent and child classes. One checkers board consists of sixty-four squares and up to thirty-two pieces. Figure 11b illustrates the GUI that was used to test the game controller before integrating it with the other subsystems. Figure 11a shows the UML diagram of the *GUI* class.

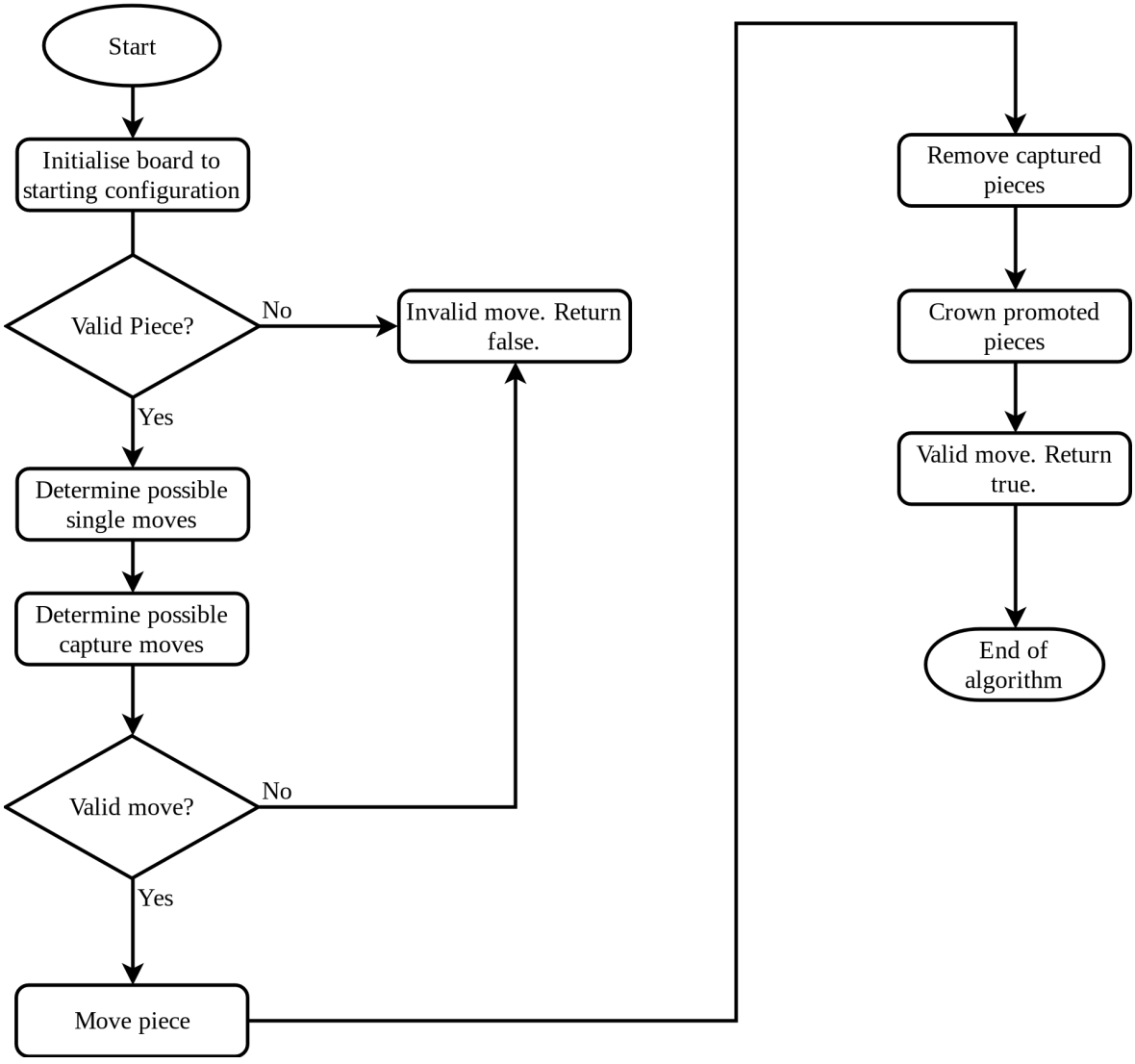


Figure 9.
Flow diagram of the *Board* class

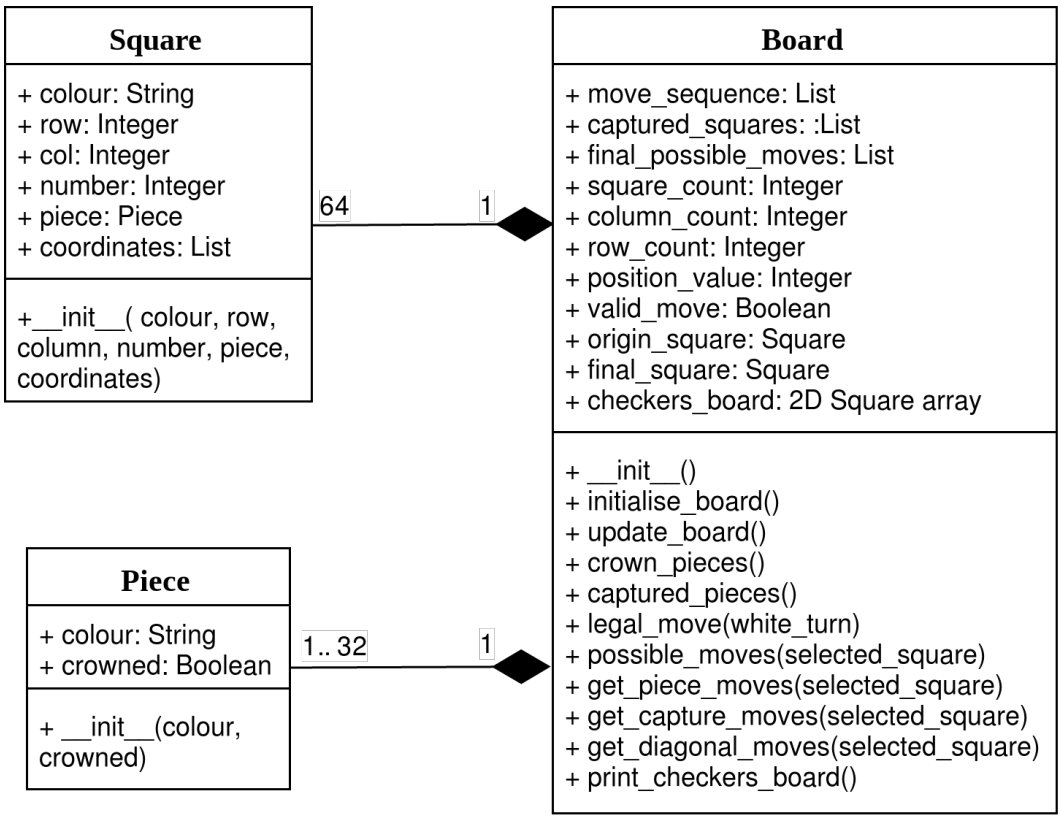
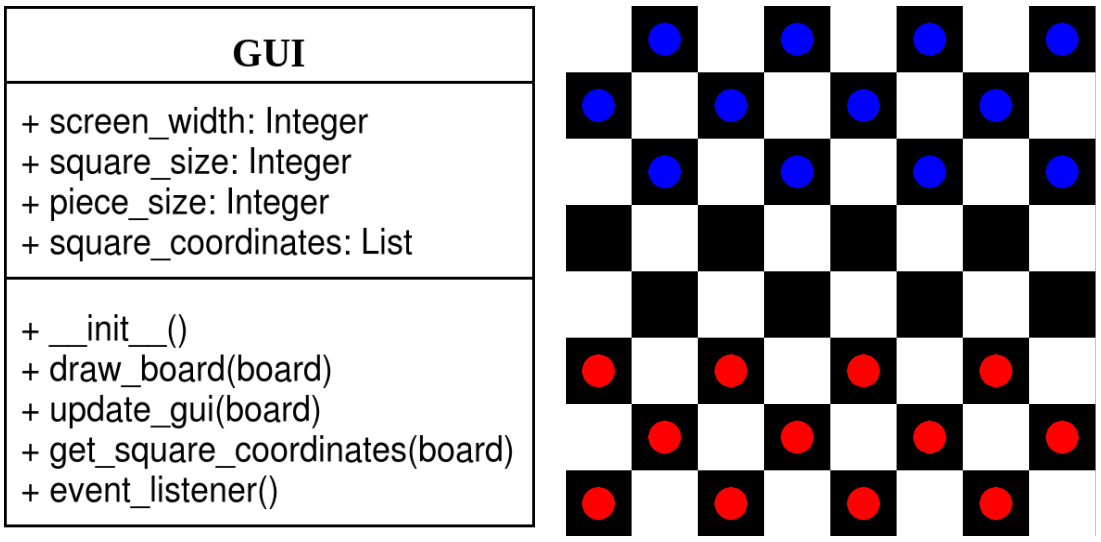


Figure 10.
UML class diagram of the board state validation subsystem



(a)
UML class diagram of the *GUI* class

(b)
Illustration of GUI used for testing

Figure 11.
GUI implementation

3.4 AI move computation

3.4.1 Software Design

Checkers is a deterministic zero-sum game of perfect information. This means that one player's gain is equivalent to the opponent's loss. Therefore, the goal of a checkers playing agent is to maximise their position and minimise the opponent's position. The possible moves of a given position can be computed and stored in the nodes of a search tree. One approach to computing the optimal move for a given position is to use the brute force approach. This method involves searching through every possible position to determine the optimal move, which is effective for games such as Tic-Tac-Toe that have a state space complexity of 10^3. Checkers has a branching factor of approximately three and state space complexity of 10^{20}, subsequently, it is infeasible to apply this approach to the game of checkers due to the excessive memory and processing prerequisites. Furthermore, this inefficient algorithm would take an inordinately long time to process moves and detract from the game playing experience.

Another approach is to apply ML to solve the problem. This allows the algorithm to essentially learn from experience rather than the traditional pre-programmed approaches. An Artificial Neural Network (ANN) models the interconnected neurons of the human brain to linearly classify inputs based on a threshold obtained from an activation function. The mathematical model for a neuron that would form part of a neural network structure taken from [11] is shown in figure 12. An example of a non-linear, differentiable activation function that computes the threshold is shown in the logistic function in equation 7. The weighted sum of the inputs for a given neuron can be calculated using equation 8, where a_i is the activation from i to j and $w_{i,j}$ is the weight of link between i and j. The neurons are interconnected in a multi-layer feed-forward configuration where the output of one neuron is the input to multiple other neurons. The ANN can be trained by feeding it training data sets and comparing the output of the final neuron to the desired outcome of a specific input. The derivative of the error is computed and back-propagated through the ANN to update the weights of the links between neurons. An ANN that employs a gradient-based optimisation algorithm can be used to map an array of inputs to the correct output.

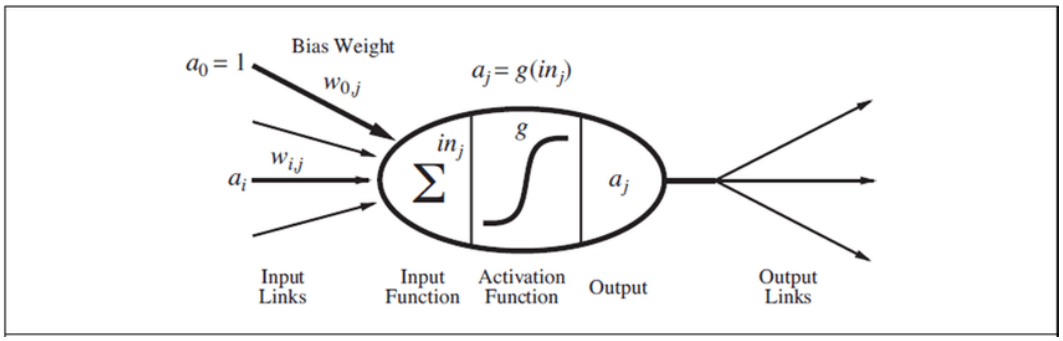


Figure 12.
Mathematical model of a neuron taken from [11]

$$Logistic(z) = \frac{1}{1+e^{-z}} \tag{7}$$

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i \tag{8}$$

Deep reinforcement learning is able to learn using a reward and punishment scheme based on the actions it takes. This is the approach employed by *Google's DeepMind* team to develop the latest (2018) *AlphaZero* chess AI [12]. A deep neural network that is trained from self-play games is used to evaluate various game states randomly generated by the Monte Carlo tree search (MCTS) algorithm. Five thousand first-generation Tensor Processing Units (TPUs) consisting of an AI accelerator application-specific integrated circuit (ASIC), generated self-play games and sixteen second-generation TPUs were used to train the neural networks in approximately nine hours. *AlphaZero* favoured positions that offered better long-term positional advantages rather than short-term material gain enabling it to win 155, draw 839 and lose just 6 games against the more traditional chess AI champion *Stockfish*. Although these results are encouraging, the budget and resources required to implement such a solution are infeasible for this project. Furthermore, checkers has a lower branching factor than chess, thus traditional techniques are able to outperform a ML approach as more game positions can be evaluated in a given time period.

In order to maximise a given position and minimise the oppenent's position, a minimax value of the specific position or node must be defined. The minimax value of a given node can be defined as the following:

$$Minimax(s) =$$

$$\begin{cases} Utility(s) & \textbf{if terminal-test(s)} \\ max_{a \in Actions(s)} Minimax(result(s,a)) & \textbf{if player(s) = max} \\ min_{a \in Actions(s)} Minimax(result(s,a)) & \textbf{if player(s) = min} \end{cases}$$

The definition assumes that the opponent plays optimally and thus maximises the worst-case situation for the player being maximised. Should the opponent play sub-optimally, the algorithm will perform even better. The minimax algorithm recursively computes the minimax values of the nodes in the search tree. Big O notation can be used to mathematically describe the relationship between the run-time of an algorithm based on the input data it processes. The minimax algorithm uses a depth first search (DFS) of the search tree to evaluate the nodes in the search tree. The time and space complexity of the DFS can be expressed using equations 9 and 10.

$$Time\ complexity = O(b^m) \tag{9}$$

$$Space\ complexity = O(b \cdot m) \tag{10}$$

b: The branching factor (number of legal moves) of the search tree.
m: The depth of the search tree.

As can be seen in equation 9, the time complexity exponentially increases as the

depth of the search tree increases, while equation 10 shows that a linear relationship exists between the space complexity and the product of the depth and branching factor of the tree. The space complexity can be improved by expanding the possible moves of the position being evaluated, rather than storing the positions of the entire game tree. As a result, the algorithm space complexity can be reduced to $O(m)$. Improvements to the time complexity of the algorithm can be achieved by implementing *Alpha-Beta pruning*, which eliminates branches of the tree that do not need to be explored. The principle of the pruning process is that if any ancestors of a node currently being evaluated have higher game state values than the current node, the current node can be pruned (removed from the search) as the game states with higher values will always be selected earlier in the game. The two parameters, alpha and beta are used to store the value of a game state while traversing the search tree:

α: Highest-value game state located along the path for the maximising player.
β: Lowest-value game state located along the path for the minimising player.

The values of alpha and beta are recursively updated and compared to the values of the current node to determine if pruning can occur. As a result, large sectors of the search tree can be eliminated, yielding a best case time complexity of $O(b^{\frac{m}{2}})$, effectively halving the overall search time of the algorithm. To take advantage of these time complexity enhancements, a move ordering function can be implemented that evaluates moves containing captures first as these are most likely to influence future game states. Other improvements include using a transposition table (hash table) to store positions that have already been evaluated, however, this requires additional memory space. Even with the *Alpha-Beta pruning* algorithm, it is infeasible to traverse the entire game tree in a reasonable amount of time. Subsequently, a heuristic evaluation function is required to replace the terminal test with a cutoff test that defines the depth of the search tree. The evaluation function returns the value of a given position based on the material value of the pieces in the position. The weighted linear function shown in equation 11 is used to mathematically express the evaluation function.

$$\begin{aligned} Evaluation(s) &= w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \\ Evaluation(s) &= \sum_{i=1}^{n} w_i f_i(s) \end{aligned} \tag{11}$$

w_i: The weight of an attribute.
f_i: The feature of the position.

In a game of checkers, a pawn could be given a value of one and a crowned piece/king could be given a value of two. The weighted linear function can be combined with other non-linear features of specific positions such as the pawns being unable to be captured when they are on the edges of the board. One of the disadvantages of the *Alpha-Beta pruning* algorithm is that it is unable to evaluate positions beyond its search depth (known as the horizon effect). As a result, in a situation where the opponent's move will result in considerable material loses for the maximising player, the maximising player attempts to delay the onset of damage by sacrificing pieces with lower material value. For example in checkers, if the maximising player's crowned piece (value of two pawns) is trapped and can be captured in the next move, a pawn may be sacrificed, as the overall material loss for the next few moves is minimised, however, the crowned piece cannot be saved and will invariably be captured by the opponent.

Fortunately, the aforementioned scenario is not that common and thus does not significantly affect the strength of the checkers AI. The block diagram depicting the design approach that was taken to solve the problem is shown in 13.

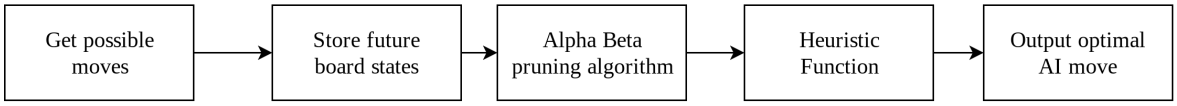


Figure 13.
Block diagram of the AI move computation subsystem

3.4.2 Software Implementation

The software for the AI move computation subsystem was developed in *Python 3.5* using *Pycharm Professional Edition 2018.3*. Figure 14 shows the UML diagram of the *AI* class. All functions were made public so that they could be accessed from the *Game* class. Variables were made private as these only needed to be accessed by their respective function. Figure 13 depicts a block diagram showing the steps that needed to be taken to compute an optimal AI move. The first step in implementing the checkers AI was to pass it the current board state (from the *Game* class) as well as other parameters such as the tree search depth and the AI's team colour. It was assumed that the AI would always play as black as the physical board could not be rotated to accommodate the user playing as black. The possible moves for each piece was acquired by creating an instance of the *Board* class and calling the *get_possible_moves* function.

The *move_ordering* function was then called, which moved the capture moves to the front of the list of all possible moves. The search tree was then constructed by first creating a deep copy of the current board state so that changes to the board could be made without affecting the current board state. The algorithm then iterated through each possible move and created a node containing the board state representation of every possible move for a given position. The copy of the board state was reinitialised to the current board state for every new possible move. The *alpha_beta* function then determined whether the maximum depth of the tree search had been attained as well as whether a terminal state had been realised. When either of these conditions were met, the *heuristic_function* was called which was able to calculate the value of a given position.

The difference between the number of opponent pieces and AI pieces was multiplied by one thousand to obtain the linear value of the position. A crowned/king piece was given a value equivalent to two pawns and positions that contained more pieces on the edges of the board were favoured over positions containing unprotected pawns. The algorithm then recursively calls itself and returns the value of alpha or beta depending on whether the game state is being maximised or minimised. The process is repeated for all of the children of the given node. If the alpha value is greater than or equal to the beta value the algorithm breaks out of the loop. The flow diagram of the AI move computation subsystem is shown in figure 15.

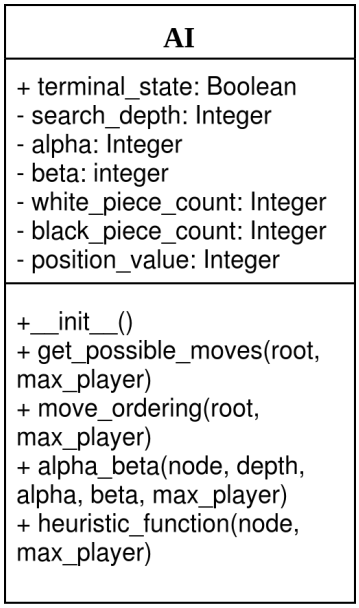


Figure 14.
UML diagram of the *AI* class

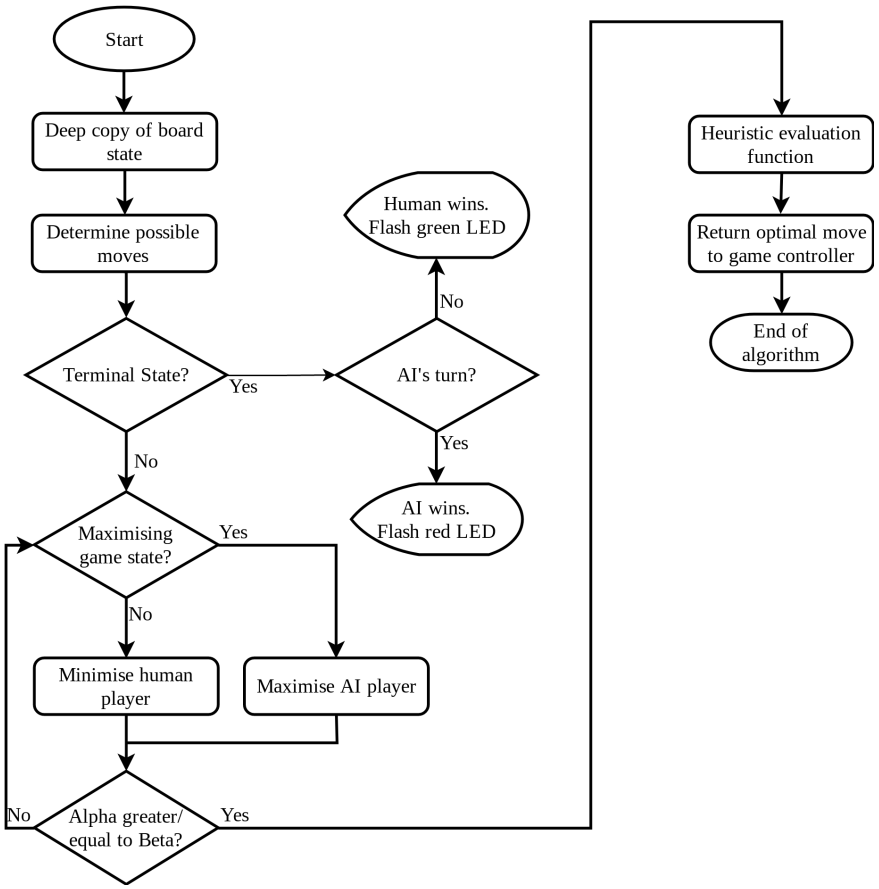


Figure 15.
Flow diagram of the AI move computation subsystem

3.5 Inverse kinematics computation

3.5.1 Software design

Following the completion of the AI move computation, the next step was to calculate how to move the robotic arm end-effector to the coordinates of a specific square. Kinematics is the study of the motion of objects without considering the external forces acting on the object. Forward kinematics is the process of calculating the Cartesian coordinates of the end-effector using the joint angles. However, in the case of a robotic manipulator, the Cartesian coordinates are known, while the vector of joint angles need to be computed. Inverse kinematics (IK) is the process of calculating the vector of joint angles given the coordinates of the end-effector. There are multiple possible solutions to the joint angle vectors due to the many DOF of a robotic arm. Multiple approaches can be used to solve the problem. One approach is to apply the Jacobian inverse technique which uses a linear approximation of the vector of joint angles. Joints are assumed to be rotational described by a single scalar value. The joint angles can be expressed as a column vector described in equation 12.

$$\theta = (\theta_1, \theta_2, \ldots, \theta_n)^T \tag{12}$$

n : Number of DOF of the robotic arm.
θ: The vector of joint angles.

The target positions are given by the vector in equation 13.

$$t = (t_1, t_2, \ldots, t_k)^T \tag{13}$$

k : Number of target positions.
t : The vector of target positions.

The desired change in the end effector position can be defined as the difference between the target position and the current position and is shown in equation 14.

$$e = t - s \tag{14}$$

e : The desired change in position of the end-effector.
t : The target position of the end-effector.
s : The current position of the end-effector.

The Jacobian matrix is defined in equation 15 and is a function of the θ values.

$$J(\theta) = \left(\frac{\partial s_i}{\partial \theta_j}\right)_{i,j} \tag{15}$$

After consideration of the aforementioned approaches it was decided to simplify the problem by breaking it down into two planes. The waist axis plane could be viewed from

above, ultimately, eliminating the other three DOF of the robotic arm. The waist axis angle could now be solved in a 2D, polar coordinate plane using trigonometric equations. First, the distance from the rotational axis to the edge of the checkers board needed to be measured. Figure 16 provides an illustration of the checkers board from a top view perspective. The distance to the board was measured to be 100 mm, however, an additional 5 mm was added to this to account for the radius of the shoulder joint.

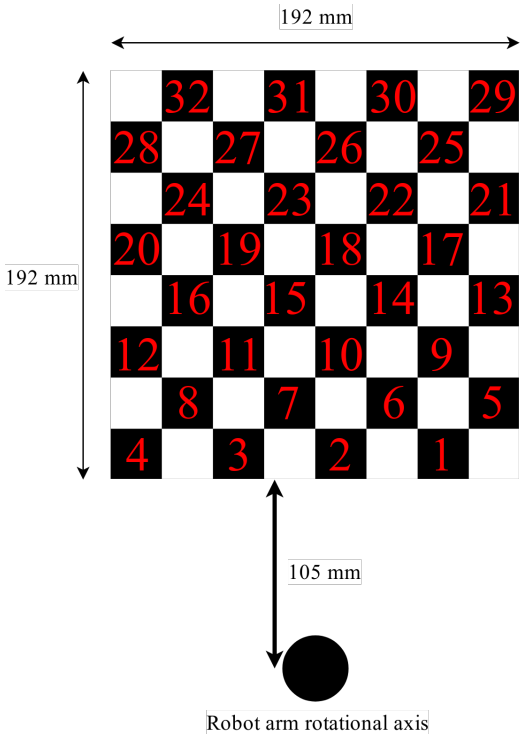


Figure 16.
Diagram showing the metrics of the checkers board

Since we know that the board is 192 mm^2, the dimensions of the squares can be calculated to be 24 mm^2. The coordinates of any square can be computed using equation 16.

$$\begin{aligned} Square\ x\ coordinate &= |3 - column| * s + \frac{s}{2} \\ Square\ y\ coordinate &= row * s + \frac{s}{2} + b \end{aligned} \tag{16}$$

s: The size of a square in mm.
b: The distance from the robot to the board in mm (105mm).

The row numbers begin with zero from the row closest to the robot arm and the column numbers begin with zero starting from the bottom left corner of the checkers board. The radius of the end-effector also needed to be subtracted from the final x-axis coordinate. After obtaining the Cartesian coordinates, the waist angle required can be calculated using equation 17.

$$\omega = \arctan\left(\frac{x}{y}\right) \tag{17}$$

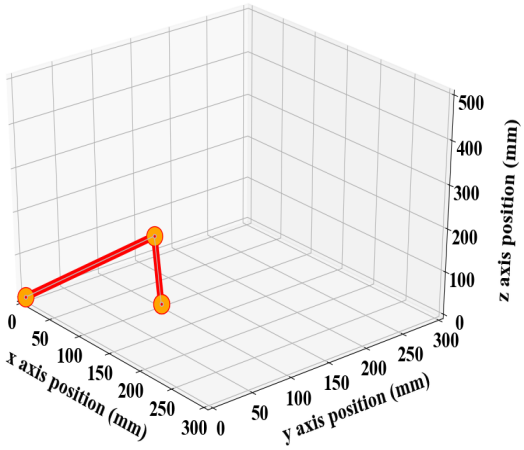
ω: The calculated waist axis angle.
x: x coordinate of the desired square.
y: y coordinate of the desired square.

The next step was to calculate the distance that the robot arm needed to travel to reach the target coordinates. This is known as the extension distance and is calculated using the theorem of Pythagoras and is shown in equation 18.

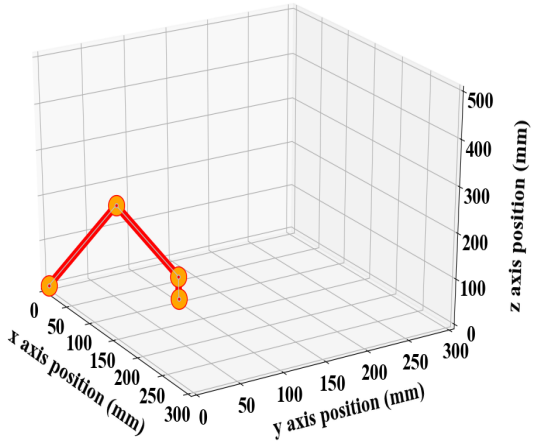
$$e = \sqrt{x^2 + y^2} \tag{18}$$

e: The required extension distance.
x: x coordinate of the desired square.
y: y coordinate of the desired square.

The shoulder, elbow and wrist angles can now be calculated using the extension distance that was calculated in equation 18. The first step in the design process was to determine the optimal lengths of the humerus (Shoulder to elbow link) and the radius (Elbow to wrist link). Simulations were conducted in *Python 3.5* to obtain a model of the robotic arm in 3D space. The goal was to create a model that was able to reach every square on the checkers board. Figures 17a shows how the problem was simplified by removing the end-effector that is shown in figure 17b.



(a)
***Python 3.5* simulation of a 4 DOF robotic arm (Without the end-effector)**

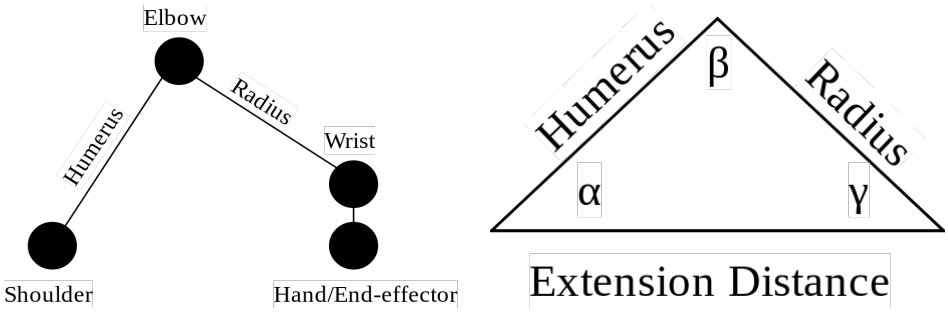


(b)
***Python 3.5* simulation of a 3 robotic arm (With the end-effector**

Figure 17.
Simulations of the robotic arm

The maximum extension distance was known to be the distance from the robotic arm waist axis to the furthest corner of the board and was calculated as 295 mm. It was found that the optimal humerus and radius lengths were 210 and 165 mm respectively. Figure 18a shows the real-world representation of the robotic arm. The problem of determining the shoulder, elbow and wrist joint angles was accomplished by simplifying the problem to the model of the robotic arm to the diagram shown in figure 18b. The end-effector was assumed to always be normal to the playing surface. The shoulder joint of the robotic arm was slightly higher

than the player surface (50mm), thus the gripper was designed to be approximately 50mm to cancel out the discrepancy and simplify the IK calculations required. This simplification allowed the IK calculations to be efficiently executed in real-time.



(a) **Side view diagram of the robotic arm**

(b) **Diagram of the simplified robotic arm**

Figure 18.
Diagrams of the robotic arm in the z plane

The extension distance, humerus and radius are all known parameters, thus the angles of the triangle can be calculated as shown in equation 19, 20 and 21.

$$\alpha = \arccos\left(\frac{h^2 + e^2 - r^2}{2 \cdot h \cdot e}\right) \tag{19}$$

$$\beta = \arccos\left(\frac{h^2 + r^2 - e^2}{2 \cdot h \cdot r}\right) \tag{20}$$

$$\gamma = 180° - \alpha - \beta \tag{21}$$

α: The angle of the shoulder joint.
β: The angle of the elbow joint in degrees.
γ: The angle of the wrist joint in degrees.
h: The length of the humerus link in mm.
r: The length of the radius link in mm.
e: The extension distance in mm.

In equations 19, 20 and 21, the cosine rule is used to obtain the angles of the simplified robotic arm shown in figure 18b. Since the joints are vertical in their home position, the waist angle is doubled to account for the 2:1 gear reduction. The final desired angles are calculated in equations 22, 23, 24 and 25.

$$Waist\ Angle = 2 \cdot \theta \tag{22}$$

$$Shoulder\ Angle = 90° - \alpha \tag{23}$$

$$Elbow\ Angle = 180° - \beta \tag{24}$$

$$Wrist\ Angle = 180^\circ - \gamma \tag{25}$$

The software design approach for the entire IK computation subsystem is depicted in the form of a flow diagram in figure 19.

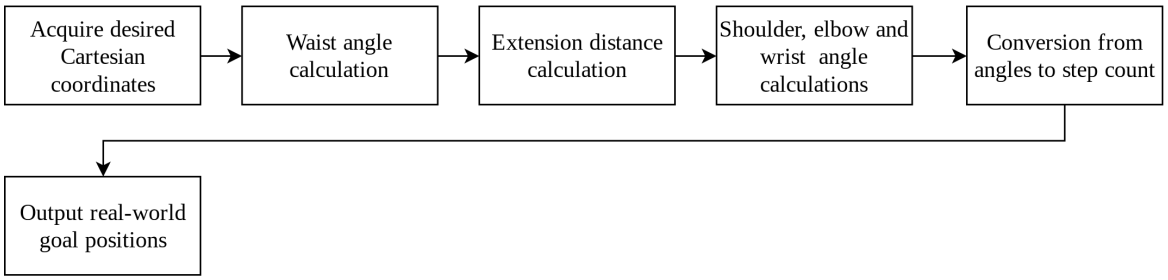


Figure 19.
Block diagram of the inverse kinematics computation subsystem

3.5.2 Software Implementation

The software for the simulations as well as the IK calculations were completed in *Python 3.5* using *Pycharm Professional Edition 2018.3*. Calculations were first verified by hand and then converted into code. Figure 20 shows the UML diagram for the *IKCalculations* class, the methods are made public, while local variable declarations are made private. The *angles_to_steps* function converts the joint angles that were previously calculated to a number of steps required to be moved by the servomotors.

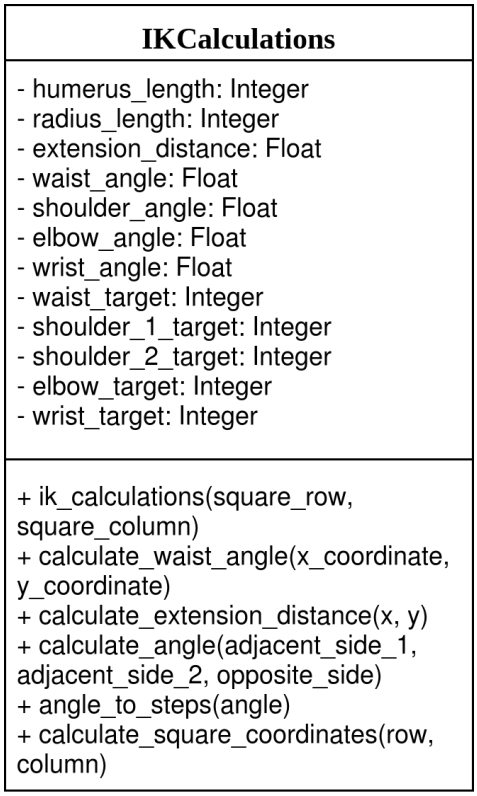


Figure 20.
UML diagram of the *IKCalculations* class

3.6 Robotic arm actuation

3.6.1 Hardware design

The final subsystem that needed to be designed was the robotic arm actuation subsystem. The system was required to actuate the move generated by the AI in the physical world. Furthermore, every square on the checkers board needed to be within reach of the robotic arm. Therefore, the first design consideration is the size of the checkers board. A full size checkers board has a square size of approximately 50 mm^2, yielding a total board size of about 400 mm^2. Thus, the reach of a robotic arm for a full size board equates to approximately half a metre. Due to budget constraints, it was infeasible to purchase actuators that would be able to handle the significant cantilever forces applied to the shoulder joint at full extension. As a result, the square size was reduced to 24 mm^2, resulting in a total board size of 192 mm^2 and a robotic arm reach of approximately 300 mm.

The robotic arm was required to provide an immersive gaming experience and as such, an articulated robotic arm design was selected as this was the least intrusive and most anthropomorphic (human-like). The next step in the design process was to determine the number of DOF that the robot arm would require. Through simulation in section 3.5 it was deduced that a minimum number of four DOF was required for the end-effector to reach every square on the checkers board. The next step in the design process was to calculate the forces that would be exerted on each joint of the robotic arm. These calculations would provide an indication of the torque requirements of the actuators. In mechanics, torque is a measure of the force that can cause an object to rotate about an axis as described mathematically in equation 26.

$$\vec{\tau} = \vec{r} \times \vec{F} \tag{26}$$

$\vec{r}$: The perpendicular distance from the pivot point to the force.
$\vec{F}$: The force acting on the object.

It is assumed that the only force acting on the robotic arm is gravity. Furthermore, the robotic arm is assumed to be at full extension, which translates to the greatest torque that the actuators will have to withstand. A safety factor of two is employed to account for the inefficiencies of the actuators as well as the effects of the angular acceleration when the robot arm is in motion. Figure 21 depicts the free body diagram that was used to calculate the torque requirements of the actuators.

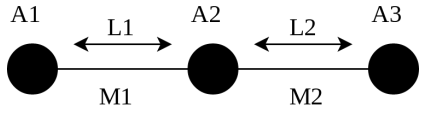


Figure 21.
Free body diagram of the robotic arm

A_1, A_2 and A_3 are the masses of the shoulder, elbow and wrist actuators and approximated to be 60g each. The waist actuator can be ignored as it is not subjected to any cantilever forces. A_1 can also be neglected as it acts a distance of zero to the pivot point. M_1 and M_2 are the masses of the links and are approximated to be 30g each. L_1 and L_2 are the lengths of the humerus and the radius and are approximated to be 200 mm each. Torque is typically measured in $N \cdot m$, subsequently, quantities were first converted to metres and kilograms and then equations 27 and 28 were used to compute the torque requirements of the elbow and shoulder actuators.

$$\begin{aligned} \vec{\tau_2} &= 2 \cdot g\left[L_2 \cdot A_3 + \frac{L_2}{2} \cdot M_2\right] \\ \vec{\tau_2} &= 2 \cdot 9.81\left[0.2 \cdot 0.06 + \frac{0.2}{2} \cdot 0.03\right] \\ \vec{\tau_2} &= 0.2943\, N \cdot m \end{aligned} \tag{27}$$

$$\begin{aligned} \vec{\tau_1} &= 2 \cdot g\left[(L_1 + L_2) \cdot A_3 + L_1 \cdot A_2 + (\frac{L_2}{2} + L1) \cdot (M_2) + \frac{L_1}{2} \cdot M_1\right] \\ \vec{\tau_1} &= 2 \cdot 9.81\Big[(0.2 + 0.2) \cdot 0.06 + 0.2 \cdot 0.06 + \\ &\quad (\frac{0.2}{2} + 0.2) \cdot (0.03) + \frac{0.2}{2} \cdot 0.03\Big] \\ \vec{\tau_1} &= 0.9418\, N \cdot m \end{aligned} \tag{28}$$

As can be seen in the torque equations, the centre of mass of the link masses is assumed to be located at the midpoint of the relevant link. As expected, the shoulder actuator is required to have the greatest torque of at least 0.94 $N \cdot m$.

Now that the structure of the robotic arm had been developed, the actuator selection could be made. To overcome the challenge of manipulating a 15 mm checkers piece, the actuators needed to have a high resolution. The resolution of an actuator is typically measured in pulses per revolution or number of steps per revolution. The resolution effectively determines the accuracy of the actuator and thus a high resolution is imperative for the design of an accurate robotic arm. The way in which the actuators are controlled is another important design decision. Closed-loop control is far superior to open-loop control systems due to the fact that the actuators operate in the real-world where undesirable influences such as backlash, pitch errors, friction and skipping of steps are prevalent. An open-loop system is unable to account for these anomalies and thus is not suitable for applications requiring a high degree of accuracy. In figure 22, a diagram illustrating how the required resolution of the robotic arm was calculated. Equation 17 was used to compute the included angle between the two target positions, assuming a maximum deviation of 2 mm of the waist axis end-effector. The calculations are shown in equations 29, 30 and 31.

$$\begin{aligned} \omega_1 &= \arctan\left(\frac{84}{285}\right) \\ \omega_1 &= 16.42218^\circ \end{aligned} \tag{29}$$

$$\omega_2 = \arctan\left(\frac{86}{285}\right)$$
$$\omega_2 = 16.79141^\circ \tag{30}$$

$$r = \omega_2 - \omega_1$$
$$r = 0.36923^\circ \; per\, step$$
$$r = 975 \; steps/revolution \tag{31}$$
$$r = 2925 \; steps/revolution(real-world)$$

r: The required resolution of the waist axis actuator.

The x and y axis coordinates of the top right square were calculated using equation 16. The top right square requires the highest resolution requirements as it is the furthest possible extension distance from the waist axis actuator. The calculated required resolution assumes perfect conditions, however, for a real-world application the required resolution has to compensate for inevitable inefficiencies. Therefore a final real-world resolution of approximately 0.1231° per step or 2924.45 steps per revolution by applying a safety factor of three.

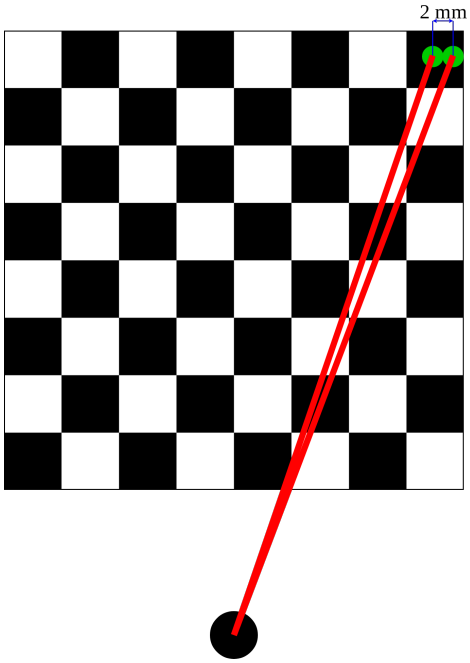


Figure 22.
Diagram used to calculate the required resolution

The first type of actuator that was considered was the stepper motor. A stepper motor uses a brushless DC motor that divides full rotation into steps. It accomplishes this by energising the magnetic teeth of the rotor. Stepper motors typically use an open-loop control system with a maximum resolution of 200 steps per revolution, which equates to 1.8 degrees per step. This resolution can be improved by using microstepping, which decreases the effective step size, however, in a real-world application, microsteps are frequently skipped,

resulting in compromised performance. Furthermore, stepper motors are unable to deliver their rated torque at the high speeds required by a robotic arm.

Servomotors contain a DC motor that is connected to a gearbox to decrease the speed and increase the output torque of the motor. Encoders are connected behind the motor to detect the current position of the motor, which is fed back to the controller that calculates the error and adjusts the motor output accordingly. Servomotors offer the torque, speed and feedback control required to develop an accurate robotic arm and thus were selected as the actuators of choice.

The next step in the design process was to select the type of servomotors required for the application. Traditional "hobby" servomotors are controlled by a 50-60 hertz Pulse Width Modulated (PWM) signal that provides the controller with the desired rotational position. The encoder transmits the current position of the motor to the controller, which calculates the difference between the target position and the current position. The first problem presented by hobby servomotors is that their rated torque does not match the actual output torque. The rated torque is generally the stall torque, which is the absolute maximum torque that the servomotor will output just before stalling.

Subsequently, the servomotor will only operate smoothly and efficiently at a torque of less than a fifth of the rated stall torque depending on the quality of the manufacturer. Similarly, the accuracy of rated resolution is also dependent on the quality of the manufacturer and delivery of rated resolution is not attained with unbranded servomotors. The second problem faced when using hobby servomotors is that, although they use a closed-loop control system, the user has no control over the parameters that define the controller outputs. These parameters are critical in controlling a servomotor that is driving a considerable amount of load that will alter the characteristics of the servomotor output. For the aforementioned reasons, hobby servomotors could not be used for the joints that had high torque and resolution requirements, such as the waist, shoulder and elbow joints.

These joints required higher calibre servomotors and thus the *Dynamixel XL430-W250-T* actuators that had a rated resolution of 4096 steps per revolution and a rated stall torque of 1.5 $N \cdot m$ were selected. Although, the rated resolution exceeded the required resolution that was calculated in equation 31, an additional 2:1 belt reduction was implemented to ultimately double the waist axis servomotor resolution. By taking a fifth of the rated stall torque, the operational torque required for the shoulder was approximated to 0.3 $N \cdot m$, which is about a third of the torque requirement calculated in equation 28. Subsequently, two *Dynamixel* servomotors were arranged in parallel to effectively double the torque output of the shoulder joint. Lightweight aluminium square tubing was used to construct the humerus and radius links. The links were made long enough so as to reduce the effective extension distance of the end-effector and thus reduce the forces exerted on the shoulder servomotors. Movement of the wrist joint was achieved using a lightweight, 9g hobby servo.

The next step in the design process was to design the hardware necessary to control the servomotors. The *Dynamixel* servomotors do not use the conventional PWM signal to control their rotational position. Instead, communication is established using a transistor-

transistor logic (TTL) multidrop bus with a half-duplex asynchronous serial communication protocol. This allows multiple servomotors to be daisy-chained on a single serial bus as each servomotor contains a unique identification number. The problem is that microcontrollers use full-duplex serial communication signals that are required to be converted into half-duplex signals to communicate with the servomotors. Half-duplex communication means that a device is unable to simultaneously transmit and receive data on a single serial bus.

One approach to solving the problem is a software-based solution, where the time required for the transmission of data is calculated and the serial bus line is driven high if transmitting and low if receiving. The fundamental issue with this approach is that status packets are sent from all servomotors, subsequently, the probability of a data collision occurring is very high. As a result, a hardware approach was ideal for solving this problem. A tri-state octal buffer was selected to drive serial bus high when transmitting data to the servomotors and low when receiving data from the servomotors. A ten kilo-ohm pull-up resistor is used to ensure a high-impedance state during power down/up. The PCB design of the servo-controller is shown in figure 23. The next step in the design process was to control

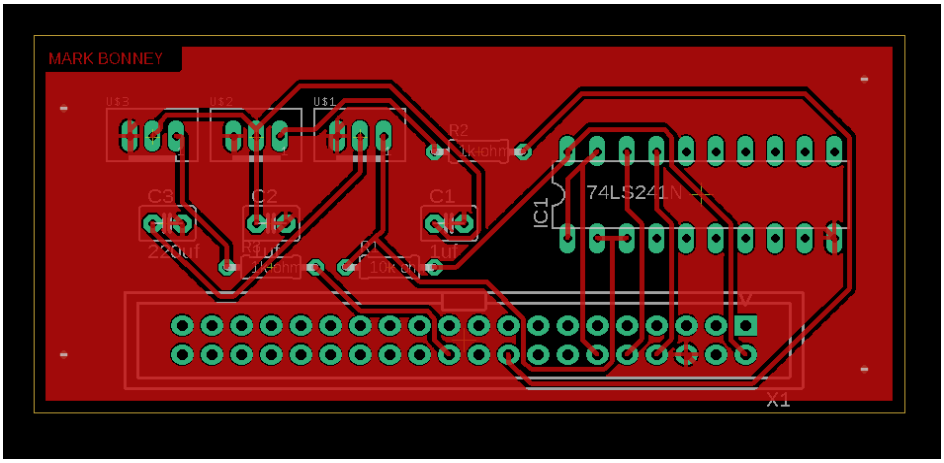


Figure 23.
PCB design of the servo-controller for the *Dynamixel* servomotors

the hobby servomotors with the use of PWM signals. A servomotor will typically centre at 1500 μs, however, every servomotor is slightly different and the horn isn't always perfectly aligned. To acquire the perfect alignment, the servomotor can be calibrated by measuring the duty cycle of the PWM signal at the operational limits. For the wrist servomotor, the duty cycle at 180 degrees was 14.99 percent and the duty cycle at 0 degrees was 3.76 percent. Using equation 32, a linear relationship between the input angle and the required output PWM duty cycle can be formulated.

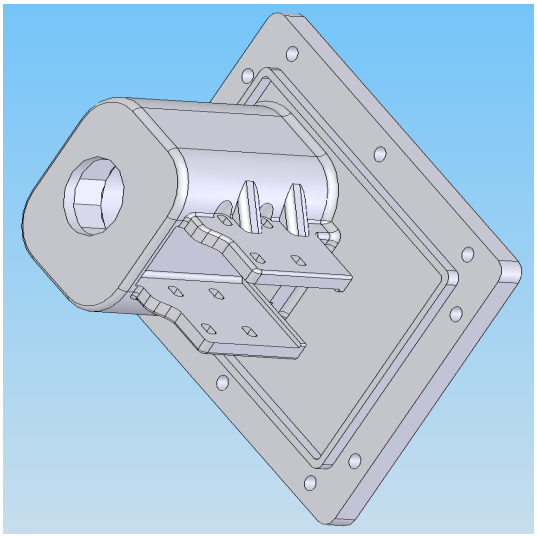
$$\begin{aligned} PWM(x) &= m \cdot x + c \\ PWM(x) &= \frac{14.99 - 3.76}{180 - 0} x + 3.76 \\ PWM(x) &= \frac{1123}{18000} x + 3.76 \end{aligned} \tag{32}$$

x: The desired angle of the servomotor.
m: The gradient of the PWM duty cycle straight line.

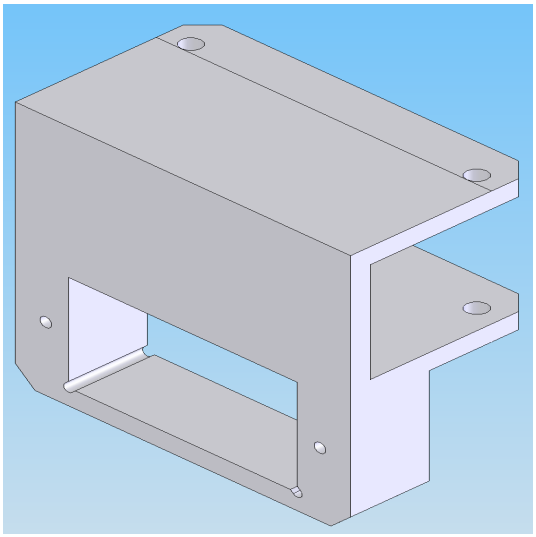
c: The y-intercept of the PWM duty cycle straight line.

Real-time performance was required to output clean PWM signals to the servomotors. The fact that *Python* periodically needed to perform memory management tasks, coupled with background kernel scheduling decisions conducted by Linux resulted in compromised real-time performance. As a result, the ATB was interfaced with a PIC microcontroller that provided the necessary real-time performance.

The final phase of the design process was to design and construct a gripper that was able to successfully manipulate the checkers pieces. Initially, an electromagnetic was considered due to its fast pick up and placement speed as well as its ability to compensate for possible errors inherent in the articulation of the robot arm. The problem encountered, was that ferrous pieces would be required and manipulating two stacked pieces would not have been possible. Traditional claw and pincer designs containing foam/rubber tips were also considered. The major drawback of these designs was that they require a repeatability and accuracy that was unattainable with the available resources. Furthermore, additional servomotors are required to control the claw mechanism, adding extra weight to the end-effector. Therefore, a suction gripper containing a bag filled with ground coffee was used to manipulate the pieces in a simple, fast and effective manner. A full size hobby servomotor was used to control the suction by adjusting the position of a syringe. The mechanical designs for the robotic arm were developed in *SolidWorks* and are shown in figures 24a and 24b.



(a)
***SolidWorks* deisgn of the waist axis servomotor reduction housing**

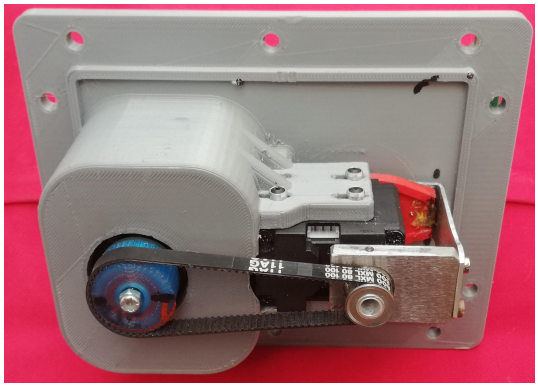


(b)
***SolidWorks* design of the wrist servomotor mounting bracket**

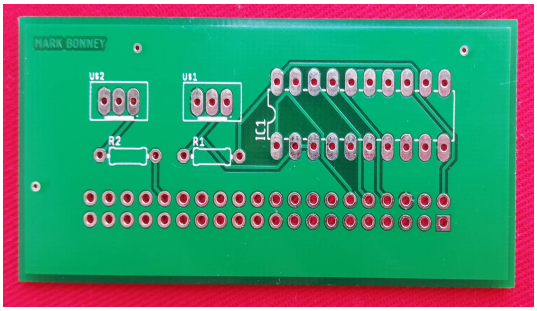
Figure 24.
Mechanical designs of the robotic arm

3.6.2 *Hardware implementation*

The implementation of the hardware designs is shown in figures 25a and 25b and operated as expected.



(a)
Implementation of the waist reduction



(b)
Implementation of the servocontroller PCB

Figure 25.
Implementation of the mechanical designs

3.6.3 *Software Design*

The block diagram of the software that was developed for the actuation of the robotic arm is provided in figure 26. Data packets were transmitted to the servomotors using a baud rate of 1Mbps, which provided high speed communication. Cyclic redundancy checks checked the incoming packets for any errors so that packets could be re-transmitted if necessary. Status packets provided important feedback from the servomotors such as error statuses, positional information, torque information and speed information. It was decided to perform synchronous transmission of data to allow all servomotors to be addressed using a single data packet. The data packet contained the target positions for each of the servomotors, PID parameters, velocity and acceleration characteristics and the respective ID of each servomotor. A step input was initially employed, however, this resulted in unstable actuation as the arm was attempting to reach the target position as quickly as possible. It was thus decided to rather use a trapezoidal profile by using a ramp input and setting the target velocity RPM to double that of the target acceleration RPM.

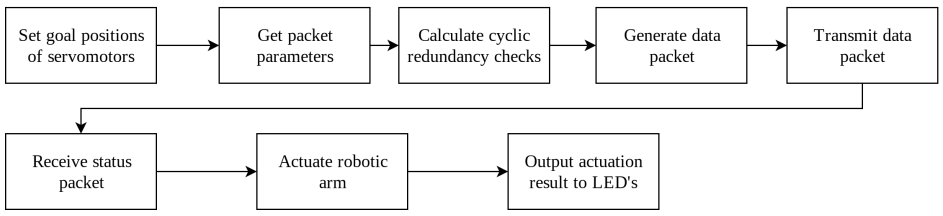


Figure 26.
Block diagram of the robotic arm actuation subsystem

3.6.4 Software Implementation

Figure 28 depicts the UML diagram of the *ServoControl* and *Registers* class. As seen in figure, the *ServoControl* is the parent of the *Registers* class, which was used to obtain the address of the command that was sent to the servomotor. Figure 27 shows the flow diagram of the algorithm that initialises the PID parameters, generates the required packets, transmits the packets and checks if the move was correctly actuated by the robotic arm by calling the board detection algorithm once the robot's turn is complete.

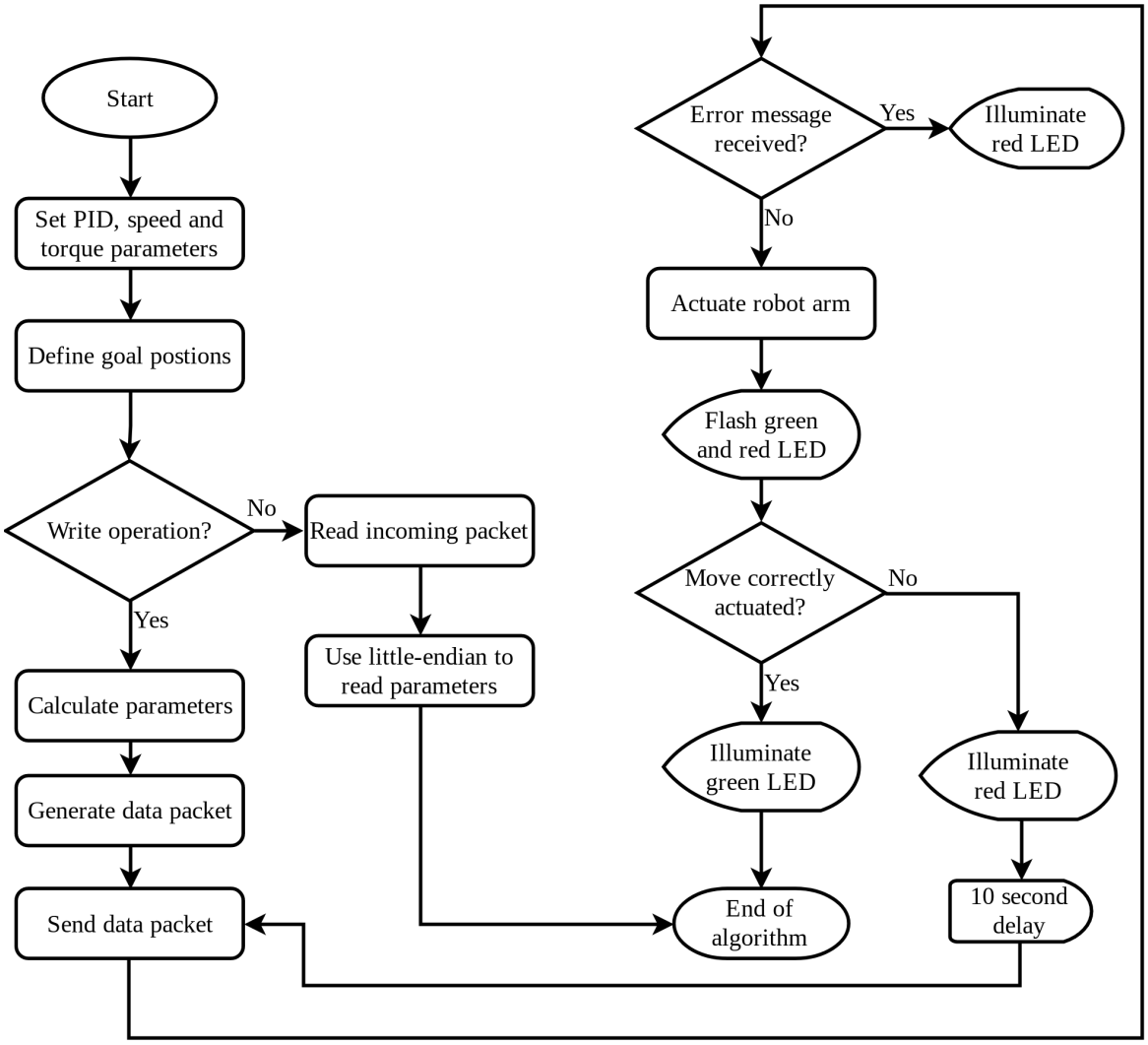


Figure 27.
Flow diagram of the robotic arm actuation subsystem

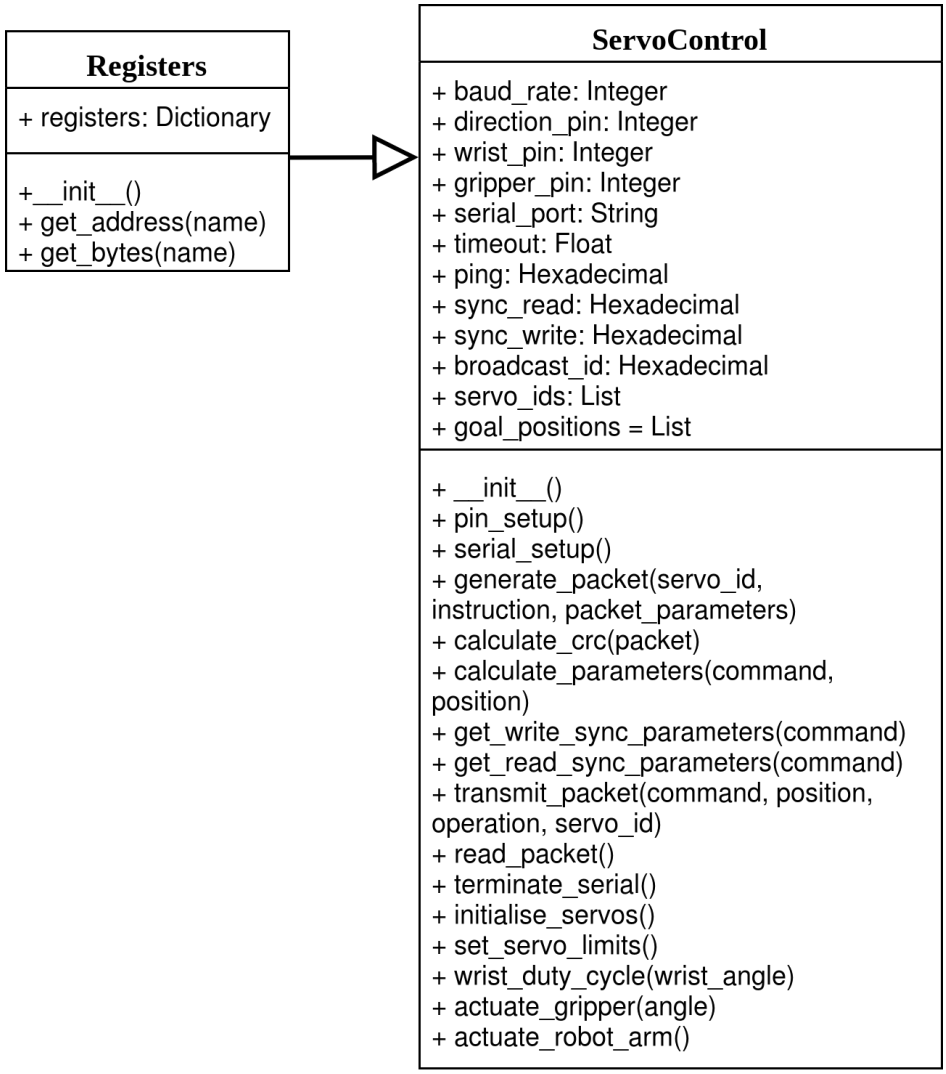


Figure 28.
UML diagram of the *ServoControl* class

3.6.5 Optimisation

A closed-loop PID was selected to control the servomotors. The mathematical description of the PID controller is provided in equation 33.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{33}$$

It was decided to employ a heuristic method known as the Ziegler-Nichols tuning method to obtain values for the Proportional, Integral and Derivative gains as it provided a quick response and a more accurate model of the plant. The ultimate gain, K_u was obtained by increasing the proportional gain until consistent oscillation was observed. The oscillation period was obtained by plotting the response in *Python 3.5* and computing the period between consistent peaks of the graph. The system was designed for no overshoot as a critically damped to overdamped response was desired. The values of K_p, K_i and K_i were obtained

using equations 34, 35 and 36.

$$K_p = \frac{K_u}{5} \tag{34}$$

$$K_i = \frac{0.4K_u}{T_u} \tag{35}$$

$$K_d = \frac{K_u T_u}{15} \tag{36}$$

K_p: Value of the proportional gain.
K_i: Value of the integral gain.
K_d: Value of the derivative gain.
K_u: Value of the ultimate controller gain.
T_u: Value of the oscillation period.

The process was repeated for the waist, shoulder and elbow servomotors. The following gains were acquired:
K_{pwaist}: 6.35
$K_{pshoulder}$: 5.21
K_{pelbow}: 6.02
K_{iwaist}: 0.1
$K_{ishoulder}$: 0.67
K_{ielbow}: 0.2
K_{dwaist}: 252
$K_{dshoulder}$: 245
K_{delbow}: 242

3.7 Integration

Figure 29 shows the full system circuit diagram. The ATB does all the CV image processing, game engine processing, AI move computation, inverse kinematics and communicates serially with the *Dynamixel* servomotors via the tri-state buffer (74LS241N). Serial communication is also established with the PIC microcontroller to ensure that the hobby servomotors operate smoothly and optimally. The PWM float values are sent to the PIC along with the protocol header of either a 'g' for gripper PWM value or 'w' for the wrist PWM value.

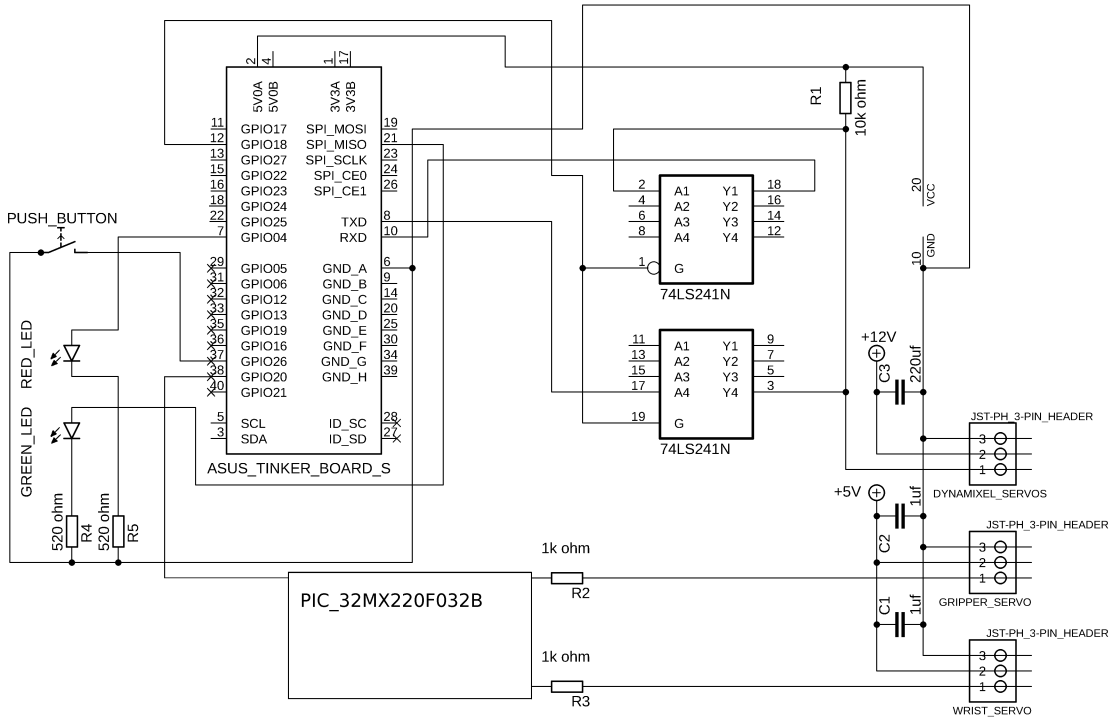


Figure 29.
Full system circuit diagram

Figure 30 shows the UML diagram for the entire system. At the heart of the software solution is the *Game* class that is the main controller of the checkers game. It is the parent of the *Board*, *AI*, *CaptureBoard*, *ServoControl* and *GUI* classes. Figure 31 shows the final implementation of the self-moving checkers robot.

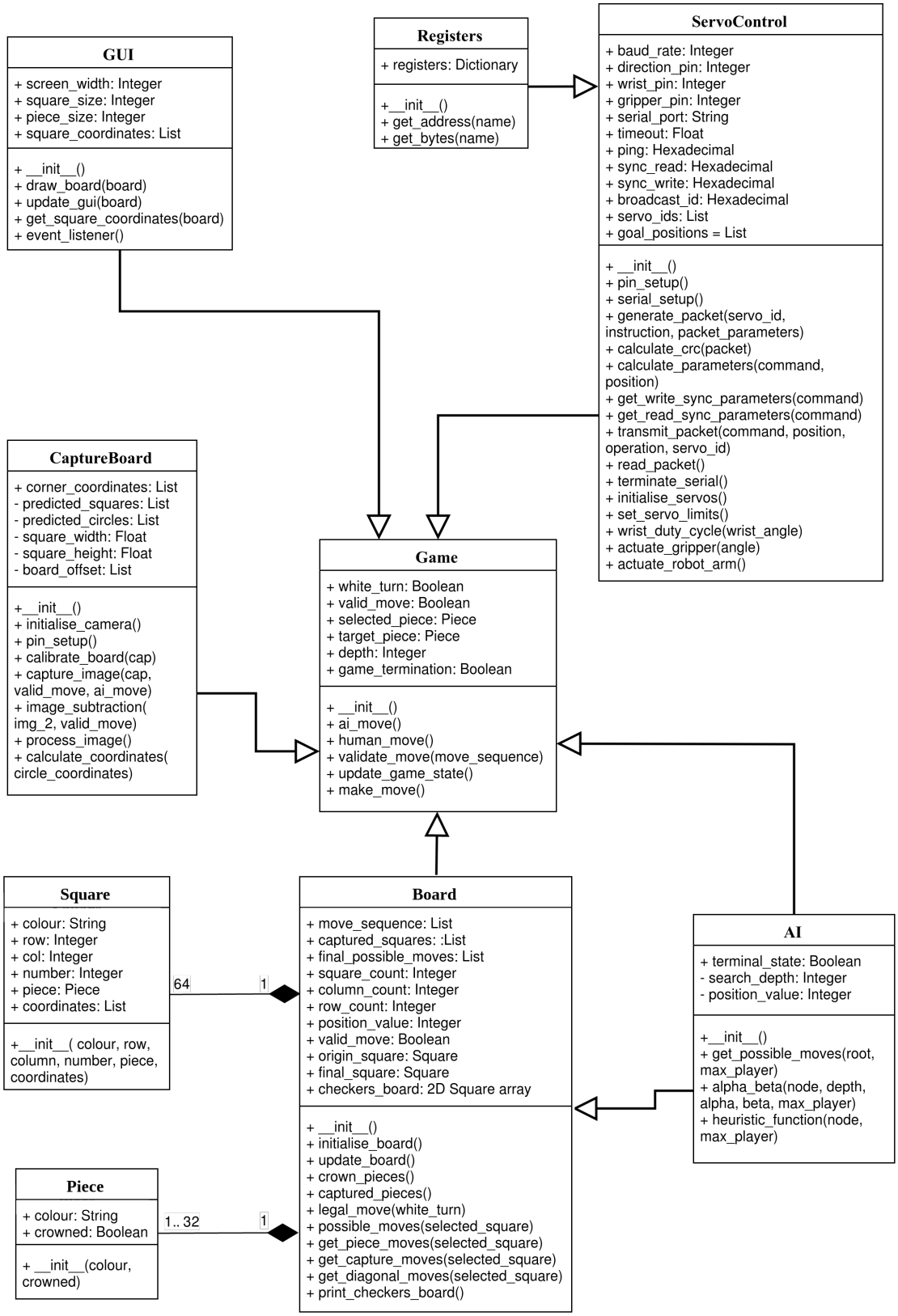


Figure 30.
UML class diagram of fully integrated system

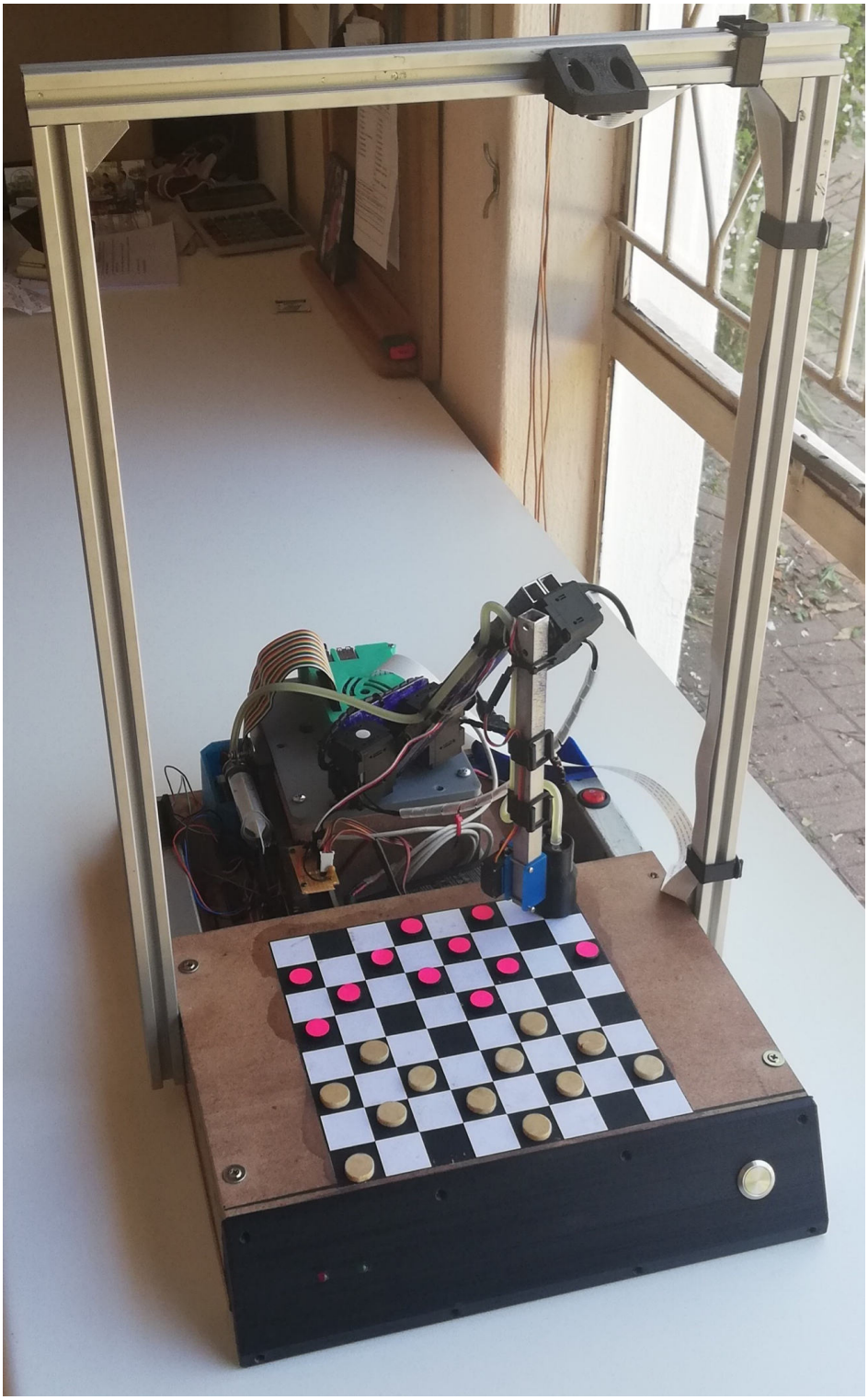


Figure 31.
Final implementation of the self-moving checkers robot

3.8 Design summary

Task	Implementation	Task completion and report section
Development and implementation of a computer vision system.	Developed in *Python 3.5*. *OpenCV* libraries were used, however, most of the code was designed and developed from first principles using *Pycharm Professional Edition 2018.3*.	Complete. Sections 3.1 and 3.2
Design and implementation of the checkers game controller and HMI.	Coded in *Python 3.5* using *Pycharm Professional Edition 2018.3*. The HMI design was completed by hand.	Complete. Section 3.3
Design and implementation of the checkers engine AI.	Developed in *Python 3.5* using *Pycharm Professional Edition 2018.3*.	Complete. Section 3.4
Design and 3D printing of the mechanical robotic arm and camera stand.	Designed in *SolidWorks* CAD software package and technical drawings were completed by hand.	Complete. Sections 3.1 and 3.6
Design and implementation of the servo controller PCB.	Designed in *Autodesk EAGLE* electronic design automation software.	Complete. Section 3.6.
Calculation and programming of inverse kinematics equations.	Developed in *Python 3.5* using *Pycharm Professional Edition 2018.3*.	Complete. Section 3.5.
Simulation of the robotic arm in 3D-space using Python.	Developed in *Python 3.5* using *Pycharm Professional Edition 2018.3*.	Complete. Section 3.6
Fine-tuning of PID control system in each servo.	Developed in *Python 3.5* using *Pycharm Professional Edition 2018.3*.	Complete. Section 3.5
Integration of hardware and software subsystems on an ARM processor.	Developed in *C* using *MPLab*	Complete. Section 3.7

Table 2.
Design summary.

4. Results

4.1 Summary of results achieved

Intended outcome	Actual outcome	Location in report
System requirements and specifications		
Once an AI move has been calculated, it should take no longer than 15 seconds to actuate a single move(Diagonal move with no hops/captures).	14.36 seconds	Section 4.2.1
The computer vision system should detect the board state and display if it is valid in less than 2 seconds.	0.43 seconds	Section 4.2.2
The AI should compute a move in no more than 10 seconds.	4.36 seconds	Section 4.2.3
Field Conditions		
The board may not move during the game. The pieces are required to be placed directly in the centre of the squares by the human player. The human player may not swap pieces in an attempt to cheat.	The system worked, provided that the board did not move and pieces were perfectly placed.	Section 4.2.2
The system is required to work indoors with artificial lighting conditions and no external lighting interference.	The system could work in consistent lighting conditions.	Section 4.2.2
Deliverables		
The design and construction of the circuitry and algorithms for the servo controller. The design and construction of the robotic arm and gripper.	The student completed the design and implementation.	Section 4.2.1
The software design and implementation required for the image preprocessing, board detection, square detection and piece detection.	The student completed the design and implementation.	Section 4.2.2
The design and implementation of the algorithm for the checkers engine from first principles. Testing of the AI against other available checkers engines.	The student completed the design and implementation.	Section 4.2.3

Table 3.
Summary of results achieved.

4.2 Qualification tests

4.2.1 Qualification test 1: Testing of move actuation

• *Objective*

The objective of the test was to test the settling times of the waist, shoulder and elbow actuators, given different target squares on the checkers board.

• *Equipment*

- *Asus Tinker Board S*
- 4 *Dynamixel XL430-W250-T* servomotors
- Physical checkers board
- Electronic storage medium

• *Test setup*

A *Python* function was created that stored the current position of the actuator every 250 μs. The position and as well as the corresponding time were stored in a text file that could be saved onto a flash drive. The robotic arm was initialised to the home position before the commencement of the experiment. The test was first executed without the use of the closed-loop control system to determine the error of the system. Thereafter, the control system parameters were set and the settling times were acquired.

• *Steps followed*

The robotic arm was first instructed to move to the furthest possible square (square 29). During the first phase of the test, the position that was acquired without the control system was compared to the target position and tabulated. The settling times of the waist, shoulder and elbow servomotors were recorded individually. A total of 2 seconds of captured data was recorded and the test was then repeated for the middle square (square 18) and finally, the closest square (square 4). The whole procedure was repeated three times to ensure that the results were repeatable. Figure 16 provides an illustration of the numbered checkers board.

The percentage overshoot is the amount that the waveform overshoots the steady-state [13] and can be calculated using equation 37. The remaining tabulated results for each servomotor are shown in tables 8, 9 and 10.

$$\%OS = \frac{c_{max} - c_{final}}{c_{final}} x100 \tag{37}$$

c_{max}: Maximum position value obtained in number of steps.
c_{final}: Set point in number of steps.
%OS: The calculated percentage overshoot.

All calculations of the *%OS* were normalised using the home value positions of the servomotors, which were 1834, 2048, 2058 and 2041 for the waist, shoulder 1, shoulder 2 and elbow servomotors respectively. The peak times were measured as the time to reach the first maximum value. Rise times were computed as the time for the waveform to go from 0.1 of the final value to 0.9 of the final value. The settling time was measured as the time taken for the transient's damped oscillations to reach and stay within 2% of the steady-state value. Computations were completed using the raw captured text file data.

• *Results*

The tabulated results of the accuracy test of the system without the control system are shown in tables 4, 5, 6 and 7.

Waist			
Target square	**Target Position (steps)**	**Position acquired (steps)**	**Error (steps)**
4	2570	2568	2
18	1758	1759	-1
29	1528	1530	-2

Table 4.
Waist axis servomotor error

Shoulder 1			
Target square	**Target Position (steps)**	**Position acquired (steps)**	**Error (steps)**
4	1731	1723	8
18	1701	1692	9
29	1570	1566	4

Table 5.
Shoulder 1 axis servomotor error

Shoulder 2			
Target square	**Target Position (steps)**	**Position acquired (steps)**	**Error (steps)**
4	2380	2393	-13
18	2416	2429	-13
29	2517	2541	-24

Table 6.
Shoulder 2 axis servomotor error

Elbow			
Target square	**Target Position (steps)**	**Position acquired (steps)**	**Error (steps)**
4	398	404	-6
18	704	704	0
29	1069	1065	4

Table 7.
Elbow axis servomotor error

The results of the PID control system are shown in figures 32,33 and 34.

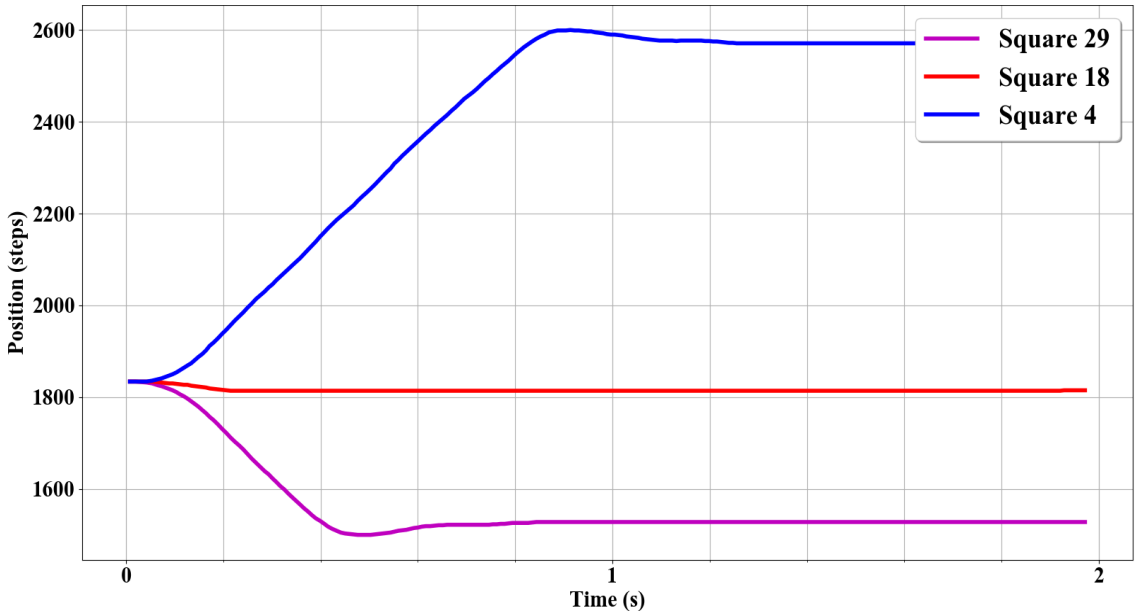


Figure 32.
Settling times of the PID controller for the waist actuator with 3 different target squares

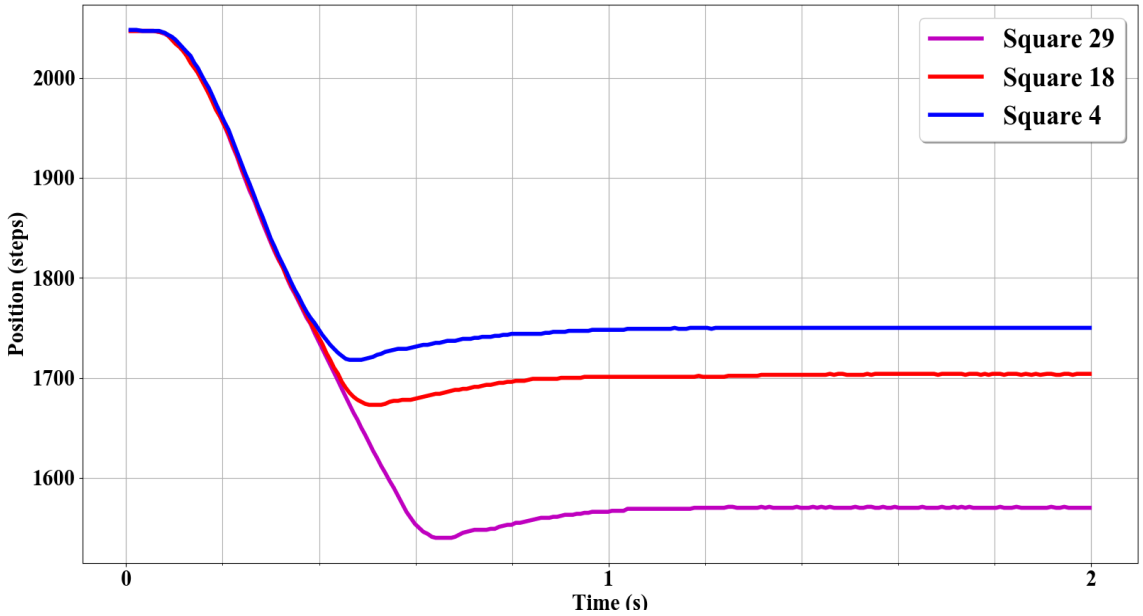


Figure 33.
Settling times of the PID controller for the shoulder actuator with 3 different target squares

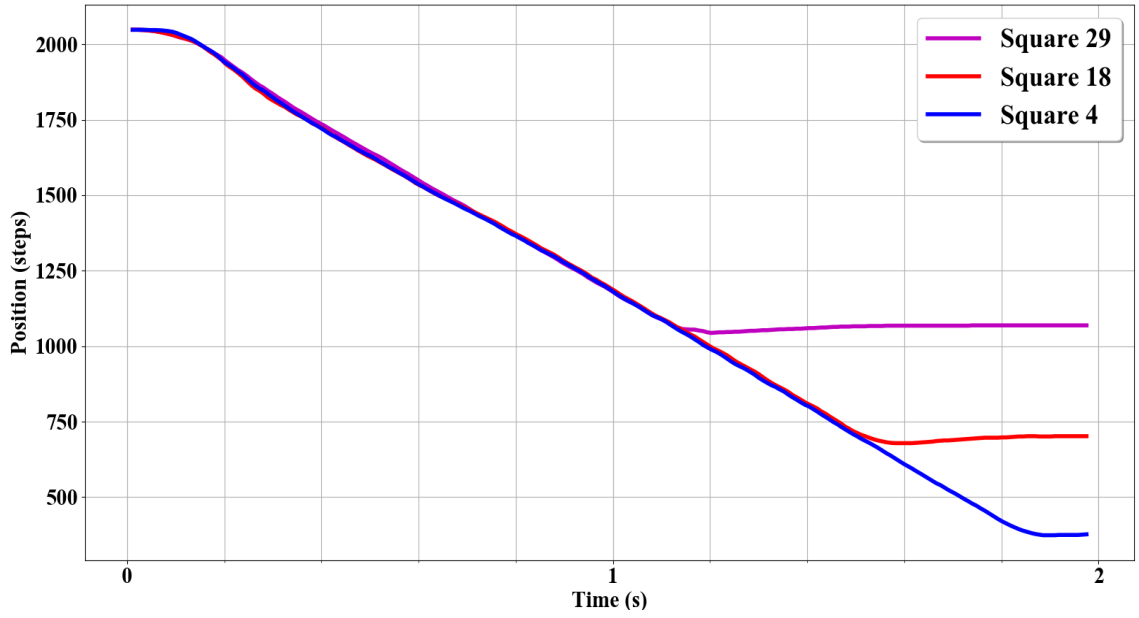


Figure 34.
Settling times of the PID controller for the elbow actuator with 3 different target squares

The PID results are shown in tables 8, 9 and 10.

Parameter	Square 4	Square 18	Square 29
Peak time (s)	0.95	N/A	0.48
Percentage overshoot	4.08	0	3.27
Rise time (s)	0.6	0.05	0.2
Settling time (s)	1.01	0.2	0.75
Steady state error	0	0	0

Table 8.
PID results of the waist servomotor

Parameter	Square 4	Square 18	Square 29
Peak time (s)	0.47	0.53	0.64
Percentage overshoot	10.73	8.09	6.28
Rise time (s)	0.21	0.22	0.35
Settling time (s)	0.92	0.83	1.01
Steady state error	0	1	0

Table 9.
PID results of shoulder 1 servomotor

Parameter	Square 4	Square 18	Square 29
Peak time (s)	1.92	1.58	1.2
Percentage overshoot	1.46	1.87	2.57
Rise time (s)	1.4	1.2	0.8
Settling time (s)	2.14	1.74	1.25
Steady state error	0	0	0

Table 10.
PID results of the elbow servomotor

A single move composed of 5 targets that are explained below:

- Target 1: The position directly above the piece that needs to be moved.
- Target 2: The z axis position of the target piece.
- Target 3: The position directly above the vacant target square.
- Target 4: The position z axis position of the vacant square.
- Target 5: The position home position of the robotic arm.

Average time to actuate a single move: 14.36 seconds
Average time to actuate a capture move: 25.12 seconds

• *Observations*

Figure 32 shows the settling times of the waist axis servomotor. The fastest settling time was observed when moving to the square 18, settling at position 1814 in 0.2 seconds, followed by square 29, settling at position 1528 in 0.75 seconds and finally the movement to square 4 yielded the slowest settling time, settling at position 2570 in 1.01 seconds. Figure 33 shows the settling times for the shoulder axis servomotor. The fastest settling time was realised when moving to square 18, settling at position 1701 in 0.83 seconds, followed by square 4, settling at position 1750 in 0.92 seconds and finally the slowest settling time was observed when moving to square 29, settling at position 1570 in 1.01 seconds. Figure 34 shows the settling times of the elbow axis servomotor. The fastest settling time was observed when moving to the square 29, settling at position 1069 in 1.25 seconds, followed by square 18, settling at position 704 in 1.74 seconds and finally the movement to square 4 yielded the slowest settling time, settling at position 398 in 2.14 seconds.

Tables 8, 9 and 10 provide a comprehensive summary of the results achieved for the various squares.

• *Statistical analysis*

Average step error for the waist servomotor: $\frac{2+1+2}{3}$ = 1.67 steps
Average step error for the first shoulder servomotor: $\frac{8+9+4}{3}$ = 7 steps
Average step error for the second shoulder servomotor: $\frac{2+1+2}{3}$ = 16.67 steps
Average step error for the elbow servomotor: $\frac{2+1+2}{3}$ = 3.33 steps

4.2.2 Qualification test 2: Testing of the board and piece detection

• *Objective*

The objective of the test was to verify the accuracy and speed of the board and piece detection.

• *Equipment*

- Physical checkers board
- 24 checkers pieces
- *Raspberry Pi NoIR V2* Camera
- *Asus Tinker Board S*
- Push button

• *Test setup*

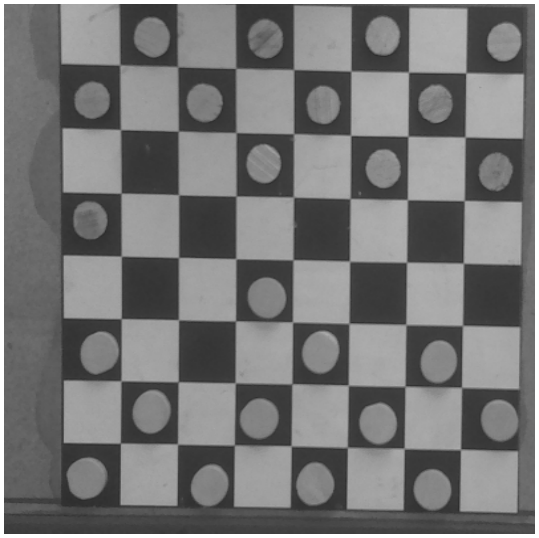
The checkers board was first calibrated and the initial configuration of a checkers game was setup. It was verified that no external lighting interference was present on the checkers board. The robotic arm actuation system was disabled. *Python's* built in timer was configured to measure the time taken for the move validation algorithm to be executed.

• *Steps followed*

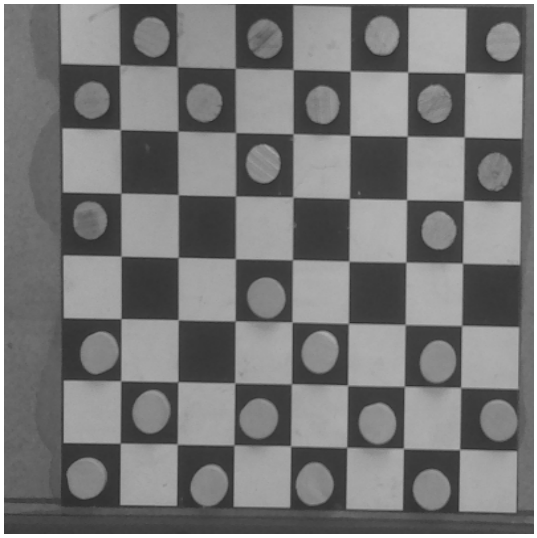
A move would be made by the student and the push button was pressed to confirm the move. The AI moves were made by the student as opposed to the robotic arm to decrease the overall testing duration. The time taken to execute the board and piece detection as well as validate the move was stored in a text file. 50 moves of a game were tracked and the average time of each move validation was computed.

• *Results*

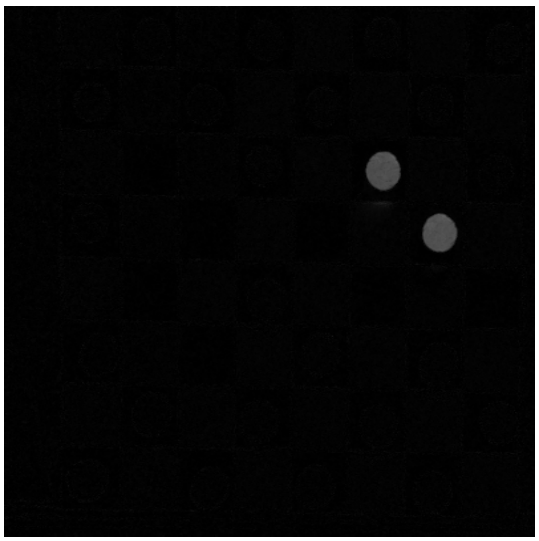
The results for the CV system are shown in figures 35, 36 and 37.



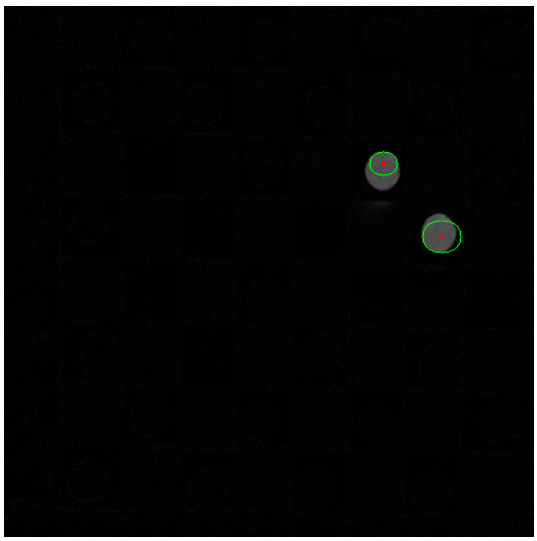
(a)
Image of previous game state



(b)
Image of current game state

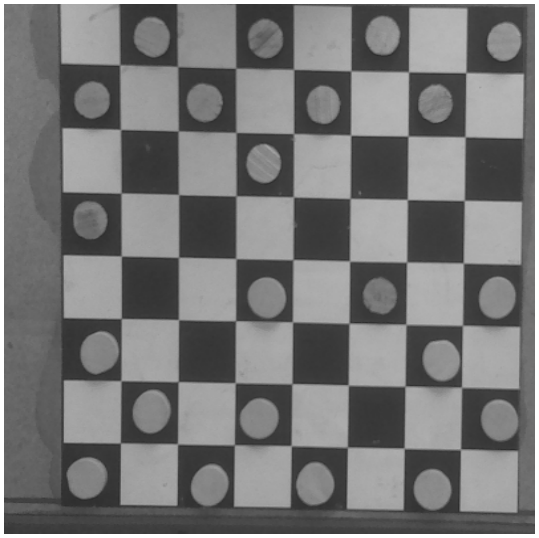


(c)
Difference image

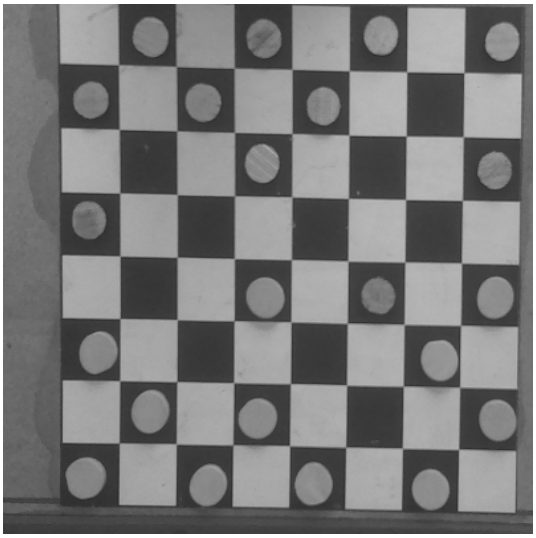


(d)
Circles detected in the image

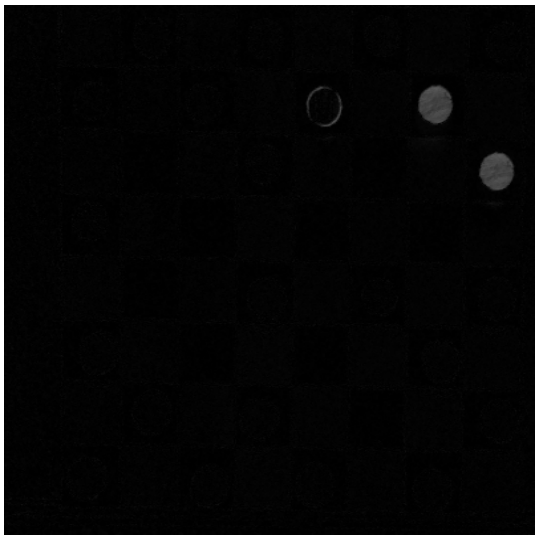
Figure 35.
Single move detected by the CV system



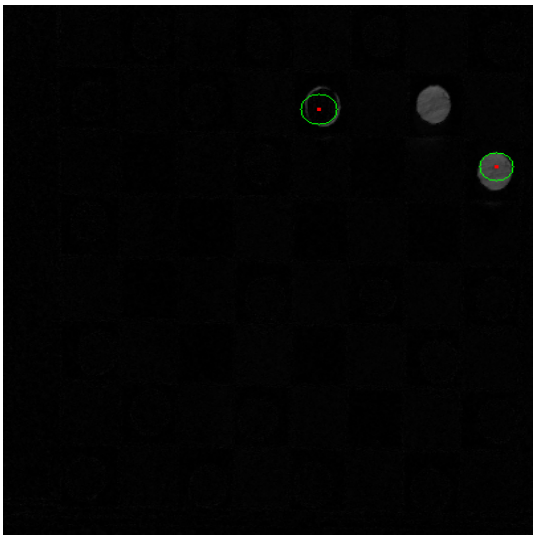
(a)
Image of previous game state



(b)
Image of current game state

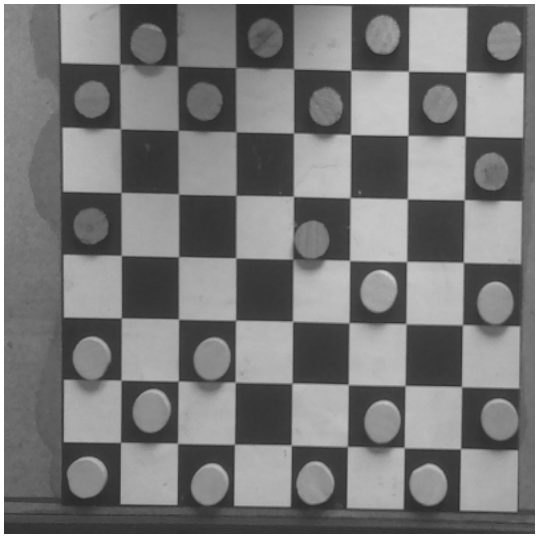


(c)
Difference image

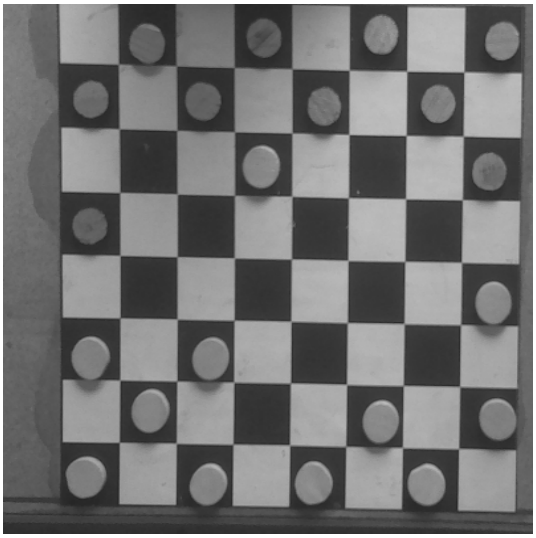


(d)
Circles detected in the image

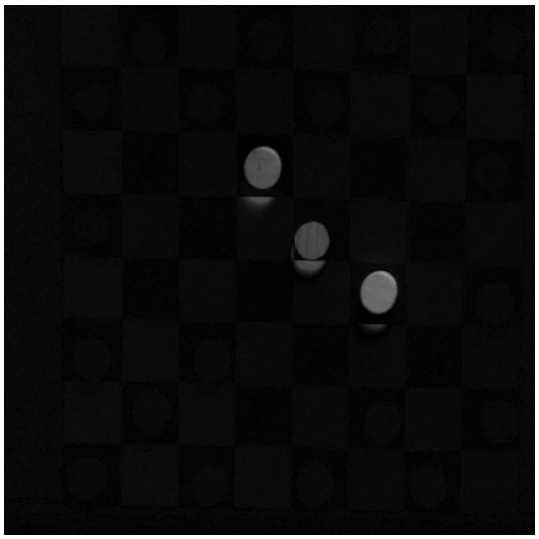
Figure 36.
The result of more than one piece moving in a move



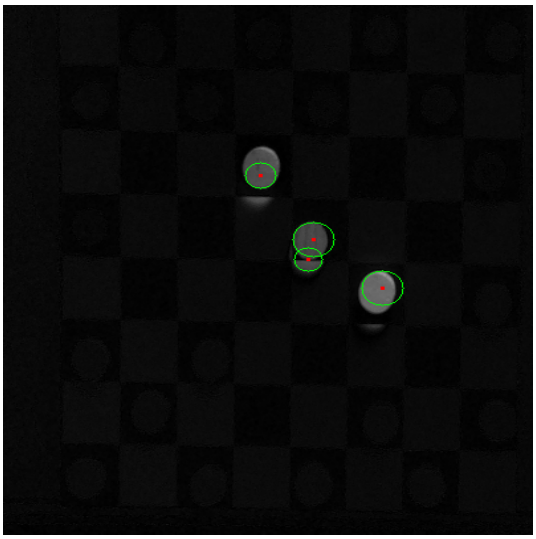
(a)
Image of previous game state



(b)
Image of current game state



(c)
Difference image



(d)
Circles detected in the image

Figure 37.
The effects of shadows in the input images

Number of moves correctly identified: 48
Average validation time: 0.45 seconds

• *Observations*

Figure 35 shows the result of the image subtraction algorithm. When the lighting conditions changed between moves, the system was unable to detect the move. When pieces that weren't involved in a move moved, they were falsely detected.

• *Statistical analysis*

48 out of the 50 moves were correctly detected, processed and validated, yielding an accuracy of 96%.

4.2.3 Qualification test 3: Testing of the checkers AI

• *Objective*

The objective of the test was to determine the speed at which the AI computed a move as well as how well it performed against another AI in a live game.

• *Equipment*

- *Asus Tinker Board S*
- *Checkers Free* mobile application

• *Test setup*

The *Checkers Free* smartphone application developed by *AI Factory* was installed on a *Huawei P20 Lite* smartphone. The self-moving checkers robot was initialised to play a game of checkers. The *Checkers Free* application was set to level 3. • *Steps followed*

The robot actuated the moves generated by the checkers AI that was developed by the student. The student then moved the moves that the *Checkers Free* AI generated. A total of 10 games were played, 5 on level 3 and another 5 on level 2. The depth of the search tree of the AI developed by the student was configured to 3 plies.

• *Results*

The results of the games played against the AI are shown in table 11.

AI performance metrics		
Performance against *Checkers Free* AI		
Level	**Wins**	**Loses**
2	4	1
3	3	2
Average time for AI move generation: 4.36 seconds		

Table 11.
Performance of the AI

• *Observations*

Out of the 5 games played against the opposition AI on level 2, 4 games were won and 1 was lost. Out of the 5 games played against the opposition AI on level 3, 3 games were won and 2 were lost. The average AI move generation time over the 10 games was 4.36 seconds.

• *Statistical analysis*

A total of 7 out of 10 games were won against the opposition AI, yielding a win rate of 70%.

5. Discussion

5.1 Interpretation of results

The accuracy of the system was first tested using an open-loop controller. This allowed for the initial error values to be recorded and interpreted. When comparing tables 7 and 5, the average step error differs by more than 6 steps. This can be attributed to the fact that the shoulder 1 servomotor requires significantly more torque to handle the high cantilever forces that its subjected to. When operating under these high stress conditions, the servomotor was unable to attain the target position. When comparing the average error obtained by the shoulder 2 servomotor in table 6 to the results obtain in table 5 a deviation of more than 8 steps was observed. The shoulder servomotors were were placed in parallel to double the effective output torque of the shoulder joint.

This should have increased the performance of both servomotors, however, the reason for this high difference was that the two shoulder servomotors were not perfectly mechanically aligned. Subsequently, the shoulder servomotor 2 had an offset of approximately 10 steps in the home position (directly vertical), yielding compromised performance. This was evident when looking at the error for the furthest extension distance of square 29, where a step error of 24 steps was observed. In table 7, a low average step error of 3.33 was observed, which is much lower than the shoulder servomotors, but higher than the waist servomotor average step error. The reason for this is that the elbow servomotor has to support the load of the wrist and end-effector weight, while the waist axis servomotor does not need to support any load. As expected, the performance of the shoulder servomotors was sub-optimal as they were required to move to the desired position while supporting the load of the weight of the elbow and wrist actuators, the gripper and the weight of the humerus and radius (aluminium links).

The results analysed above, indicate that open-loop control of the robotic arm would have been incapable of achieving the required accuracy. As a result, a closed-loop control system was required. Each servomotor had to be individually fine-tuned to offer satisfactory results. Figure 32 illustrates the closed-loop control system that was implemented for the waist servomotor. A detailed description of the results for the waist servomotor are shown in 8. As expected, the fastest settling time of 0.2 seconds was observed when moving to the middle square as very little positional displacement from the central home position was required. The second fastest settling time of 0.75 seconds was observed when moving to the furthest square (square 29), this was faster than moving to the closest square (square 4) for two reasons. The first reason was that the closest square exhibited the greatest positional displacement required from the home position, due to the largest included angle between the central home position and square 4. This can be seen in the large step size from home position of 1834 steps to square 4's position of 2570 steps. The second reason for the large discrepancy between the positional displacement of square 29 and square 4 (other than the angle) was that the waist axis was not perfectly mechanically centred on the midpoint of the checkers board. As a result, moving to squares on the right hand side of the checkers board yielded a marginally faster settling time of the waist servomotor, when compared to moving to the symmetric counterpart on the left hand side of the checkers board. The closed-loop control system was designed to be critically damped to overdamped, which would yield a

response that was free of overshoot. A ramp input was selected over a step input to provide a constant velocity and an overall smoother response. When implementing the designed values in practice, it was found that the settling time of the servomotor was greater than 3 seconds and thus the integral component of the control system had to be increased by a factor of 2 (from 500 to 1000) to reduce the settling time. It was noted that the movement to square 18 resulted in a response free of overshoot, which can be explained by the minimal size of the error between the home position and the target position. The sacrifice of a slight overshoot of 4.08% for square 4 and 3.27% for square 29, yielded a significantly reduced settling time which was required in order to meet the time specifications of the move actuation. It was observed that a relationship between the step size (target position minus initial position) existed, such that the greater the step size, the greater the overshoot that was observed. A faster rise time could have been achieved by increasing the velocity of the ramp input, at the cost of a greater average overshoot, ultimately increasing the settling time. Furthermore, the movement of the robotic arm was required to be smooth and controlled, thus an erratic response with a percentage overshoot greater than 15 would have yielded unsatisfactory results.

Figure 33, depicts the settling times for the shoulder servomotor. It is important to note that due to the fact that the two shoulder servomotors were mechanically connected to the same link, a single closed-loop control system was implemented to control the shoulder joint. Initially, it was attempted to implement closed-loop control for both servomotors, however, this resulted in the overheating of the second shoulder servomotor that was mechanically offset from the home position. The reason for the overheating was that the closed-loop control systems of each shoulder servomotor were effectively working against each other. The steady state error correction of one servomotor was counteracted by the second servomotor. As a result, the second shoulder servo motor was controlled using an open-loop configuration, while the first shoulder servomotor was controlled by means of a closed-loop control system.

Therefore the results given in figure 33 and table 9 are indicative of the first shoulder servomotor's performance, that was effectively controlling the entire shoulder joint. Figure 33, shows that the settling time increased as the required extension distance of the robotic arm increased. This can be expected as the positional displacement required to reach further is increased and thus the time taken to move to the target increased. When comparing the percentage overshoot of the various squares, it is interesting to note that the highest overshoot of 10.73% was observed when moving to square 4, compared to the 6.29% recorded for square 29. This was unexpected as the step size of 298 steps to reach square 4 is much smaller than the 478 steps to traverse to square 29. The reason for this anomaly may be due to the fact that a greater step size of the waist and elbow servomotors was required to align the robotic arm with square 4, subsequently, greater moments of inertia were experienced by the shoulder servomotors. Another reason could be that the shoulder joint accuracy was compromised by the second shoulder servomotor that used an open-loop control system. This could also explain the steady state error of 1 step that was observed when moving to square 18.

Figure 10 shows the settling times of the elbow servomotor. As expected, the lowest settling time was observed when moving to the furthest square as it had the smallest step size. It was noted that the rise time of the response could have been improved by increasing the ramp input velocity for the elbow servomotor. However, although this

may not have affected the performance of the elbow servomotor response, it may have degraded the performance of the shoulder servomotors due to the increased angular acceleration of the end-effector. As was the case with the shoulder servomotor, the percentage overshoot of the system was inversely proportional to the step size. Again, this unexpected anomaly could be attributed to the fact that all three control systems are operating in parallel in order to simultaneously actuate all servomotors and reduce overall actuation time.

The average time to taken to actuate a single move of 14.36 seconds was satisfactory as it met the requirement of being below 15 seconds. The time taken to execute a capture move was significantly higher than a single move, as the robotic arm needed to first execute the capture and then proceed to remove the captured piece from the board. The robotic arm was required to move to intermediary targets, manipulate the piece and return to the home position to execute a single move, subsequently, the total move time is significantly higher than the time taken to settle at a single target. The design decision to use the universal gripper as opposed to a traditional claw, significantly improved the speed and reliability of the move completion. The increase in speed performance was due to the reduction in the overall weight of the end-effector as the servomotor controlling the syringe for the suction mechanism could be placed at the base of the robotic arm.

Conversely, a traditional claw design would have required an additional servomotor placed on the end-effector as well as additional closed-loop feedback to determine whether a piece had been grasped. The reliability of the piece manipulation was also improved as the universal gripper offered a higher margin of error in the z axis. The bag filled with coffee grounds could be pressed into the surface of the checkers board and conform to the shape of the checkers piece. Conversely, a claw mechanism would have required the z axis height to be accurate to within a mm of the target height and as a result would not have been as reliable as the universal gripper. Finally, the gripping strength of the universal gripper far exceeded that of a traditional claw as was demonstrated in [10].

Although the universal gripper significantly improved the overall reliability of the system, manipulation errors did still occasionally occur. If for example, a checkers piece was not perfectly placed in the centre of a square at the commencement of the game, the error would propagate to the future moves involving the imperfectly placed piece.

The average time taken to validate a move was 0.43 seconds which was satisfactory as it was below the minimum requirement of 2 seconds. Furthermore, the design decision to use a hardware trigger for the move completion, allowed the game to be played at a rapid pace and improved the overall efficiency of the algorithm as continuous processing of images was not required. An accuracy of 96% for the board and piece detection was achieved. This allowed a checkers game to be tracked, provided that no external lighting interference was present. In figure 35, a single move input is appropriately detected as two circles. The algorithm was able to successfully locate the coordinates of the centre of the circles. Figure 36 shows the result of more than a single piece moving in a move. Figures 36a and 36b show the piece in the second row of the seventh column moving to the third row of the eighth column. The piece in the first row of the fifth column also moves and this can be seen in centre of figure 36c. The outline of the circle is detected in figure 36d, while the desired piece in row two,

column seven remains undetected. This meant that only the piece being moved as well as the captured piece could be moved in a player's turn. Figure 37 shows a board state in which many shadows are present as a result of an external light source. In figures 37a and 37b a capture move involving three pieces is depicted. A shadow formed by the captured piece overlaps with another square as is seen in the middle circle of figure 37c. Consequently, in figure 37d it is seen that this shadow is falsely detected by the *HoughCircles* algorithm. The solution to this problem was to ensure that the light sources were placed directly above the checkers board.

The AI system was successfully able to generate an appropriate move in 4.36 seconds which was satisfactory as it was within the 10 second limit. A win rate of 70% was achieved against an off-the-shelf AI checkers engine which was acceptable given the budget and time constraints. It was noted that a logarithmic relationship existed between the search time and the depth of the search tree. It was determined that a search tree with a depth of 3 plies was found to provide an appropriate move in a reasonable time period. The *Alpha-Beta pruning* algorithm was consistently able to generate the optimal move for a given position. The checkers engine struggled in situations where a search depth greater than 3 plies was required to determine the optimal move. Furthermore, it struggled in end game situations where a significantly higher number of possible moves are available. As such it would prefer to select passing moves to conserve material rather than devise an intuitive strategy for victory.

5.2 Critical evaluation of the design

Aspects to be improved in the present design

The performance of the shoulder servomotors could be improved by using a planetary gearbox reduction that would significantly decrease the settling time of the shoulder servomotor. Furthermore, it would enable a single shoulder servomotor to be used which would eliminate inefficiencies present in the shoulder joint control system. Alternatively, a counter-balance that rotates in the opposite direction to the end-effector could be used to improve the performance. Given more funding, actuators with a higher resolution could be purchased that would enable faster move actuation. The system mechanics could be improved by laser cutting lightweight aluminium parts specifically designed for a robotic arm as opposed to using aluminium tubing.

The system could be modelled in an off-the-shelf 3D simulator that contains a physics engine to account for aspects such as angular acceleration and frictional forces. The specific servomotor inefficiencies could be incorporated into the design with more specific details as opposed to using safety factors. Real-time visual feedback could be obtained with the use of a camera mounted in the gripper as well as another camera to provide depth perception of the physical world. The movement of the entire system could be modelled as a closed-loop control system to provide faster settling times. Given more time, this could be combined with ML approaches to optimise the error of the entire system.

The CV system could be improved by continuously processing image data to detect movement on the checkers board. Ultimately with a more powerful processor, hand detection could be used to detect the completion of human moves without the use of a hardware trigger. False detection of circles could be reduced by processing detected circles sequentially instead

of as an entire move.

The AI engine could be improved by increasing the effective search depth of the search tree. This could be achieved by using a more powerful processor. The heuristic evaluation function could be improved given more time and funding by incorporating ML techniques that could provide enhanced reasoning for end-game positions.

Strong points of the current design

The robotic arm was able to efficiently and effectively move to every square on the checkers board. The design of the closed-loop control system was able to reduce the steady-state error and account for an array of real-world inefficiencies. The gripper design provided a low-cost, lightweight and effective solution to the challenge of manipulating 15mm checkers pieces. Furthermore, the challenge of picking up two checkers pieces stacked on top of each other was achieved, provided that the pieces were perfectly placed. The gripper design meant that checkers pieces were not required to be magnetic as was the case with an electromagnet. The design of a responsive CV system allowed changes in the board state to be detected in real-time and with minimal processing power.

Failure modes and effects analysis

The CV system fails when the lighting conditions between moves changes or if a piece moves that is not involved in a move. Should any of these two conditions occur, the CV system detects the move as invalid even if it was valid. The move is then required to be re-entered by the user for the game to continue.

The move actuation may fail if the checkers pieces are not placed in the centre of the squares. If the robotic arm is unable to actuate the move, human intervention is required to place the checkers piece in the correct position to allow the game to continue.

The AI checkers engine was unable to detect double captures (multiple hops) as it was unable to think more than 3 plies ahead.

5.3 Design ergonomics

The system was designed to provide an enjoyable user experience. The design and development of a user-friendly front panel provided the user with critical feedback on move validation in the form of green and red LED's. The panel was designed with a 45 degree angle to ensure that the outputs and inputs were easily noticeable and accessible. A large stainless steel push button was installed that enabled the user to confirm their move. An intuitive power switch along with a kettle mains plug was developed to enable the device to be power directly from a mains socket. A compartment for holding checkers pieces was installed at the back of the unit. All of the power supplies were positioned under the board to that enhanced the aesthetics of the product.

5.4 Health, safety and environmental impact

The system contains an emergency stop button that shuts down the robotic arm in the event of a potential collision. The servomotors also are designed with position limits to ensure that it does not move outside of the realm of checkers board playing surface. Furthermore, the servomotors have torque limits that shutdown the servomotor if exceeded.

The product is designed to be modular and thus parts can easily be replaced if necessary. The electronic components cannot be recycled and must be safely disposed when it reaches the end of its lifetime. The system draws a low amount of current that does not exceed 2A at maximum load.

5.5 Social and legal impact of the design

The product has many real-world applications such as manufacturing and automation. The robotic arm is able to complete repetitive tasks easily and effectively and thus could replace human workers in many different working environments. This would save business money, however, at the cost of an increased unemployment rate. It also has the ability to assist the physically handicapped in performing tasks in the home environment. Thorough testing and development must be executed prior to employing robots both in the workplace and homes.

6. Conclusion

6.1 Summary of the work completed

This report describes the design and development of a self-moving checkers robot, capable of playing an immersive game of checkers against a human opponent.

A literature study of the existing chess and checkers robot implementations assisted in acquiring an understanding of the challenges inherent in the design process. Equipped with this knowledge, a vision system capable of detecting changes in the board state was designed and developed. Validation of the detected moves was accomplished by designing and developing a checkers game engine. The generation of an appropriate response to the user move was achieved by designing and implementing a checkers AI. The move produced by the checkers AI was actuated in the physical world by designing and developing a 4 DOF articulated robotic arm. The derivation of mathematical models representing the physical checkers board allowed the robotic arm to move to any square on the checkers board. A gripper that comprised of a bag filled with coffee grounds attached to a suction device was engineered to manipulate the checkers pieces. The successful integration of all of the subsystems resulted in an average single move actuation time of 14.36 seconds.

6.2 Summary of the observations and findings

The self-moving checkers robot was able to interpret a move in 0.43 seconds, generate a move in 4.36 seconds and actuate a single move in 14.36 seconds. Figure 33, illustrates the exponential relationship between the distance of a square from the robotic arm and the settling time. Figure 35, illustrates the processing and interpretation of changes in the game state. The quality and efficiency of the computer vision algorithms were directly proportional to quality of the input images. Finally, the processing time of the AI move computation was on average halved by employing the *Alpha-Beta pruning* algorithm which significantly reduced the size of the search tree.

6.3 Contribution

Many new mechatronics and CV principles such as inverse kinematics and movement detection were required to be mastered in order to successfully complete the project. Software packages that needed to be mastered included *Autodesk EAGLE* electronic design automation software to design the PCB for the servocontroller. Understanding of hardware devices such as a single board computer for all processing and servomotors that consisted of a unique communication protocol. The integration of all software and hardware into a standalone prototype required every subsystem to be operating optimally. *SolidWorks* and 3D printing principles needed to be mastered in order to manufacture robust and reliable hardware components.

A unique approach to engineering the CV system that combined calibration and image subtraction techniques. *OpenCV* functions were used to process the input images. An object-oriented approach was taken to develop a checkers engine from first principles. The *C* code for the PIC microcontroller was developed from first principles. The study leader assisted in providing context to subsystems that needed to be developed.

6.4 Future work

The processing of moves was compromised by changes in lighting conditions between moves. Future developments could implement algorithms that account for the detection of false positives and decrease the noise in the input images.

Given more time and funding, higher quality actuators could be purchased that would enable the use of a full size board. Furthermore, mechanical inefficiencies could be reduced by using a planetary gearbox for the shoulder axis servomotor and a belt reduction of the elbow servomotor to reduce the overall weight of the robotic arm.

The accuracy of the robotic arm could have been improved by adding real-time visual feedback to the servomotor controller. This would require an additional camera for depth perception. ML approaches to improve accuracy could have been explored, given more time.

The simulation models used did not accurately account for all real-world inefficiencies such as moments of inertia. An off-the-shelf simulation software package could have been used to improve the simulated models of the system.

The strength of the AI could have been improved by using a processor capable of creating a larger search tree. Given more time, ML approaches to improve the output of the heuristic evaluation function.

In conclusion, in the future, a more intelligent robot that is able to learn from its mistakes and evolve could be developed.

7. References

[1] H. M. Luqman and M. Zaffar, "Chess brain and autonomous chess playing robotic system," *IEEE*, pp. 211–216, December 2016.

[2] E. Sokic and M. Ahic-Djokic, "Chess brain and autonomous chess playing robotic system," *IEEE*, pp. 75–79, February 2009.

[3] R. A. M. D. L. B. R. C. M. K. L. L. J. R. S. Cynthia Matuszek, Brian Mayton and D. Fox, "Gambit: An autonomous chess-playing robotic system," *IEEE*, pp. 4291–4297, August 2011.

[4] T. M. K. Dennis Aprilla Christie and P. Musa, "Chess piece movement detection and tracking, a vision system framework for autonomous chess playing robot," *IEEE*, February 2018.

[5] N. Banerjee, "A simple autonomous robotic manipulator for playing chess against any opponent in real time," *International Conference on Computational Vision and Robotics*, August 2012.

[6] A. T.-Y. Chen and K. I.-K. Wang, "Computer vision based chess playing capabilities for the baxter humanoid robot," *IEEE*, pp. 11–14, June 2016.

[7] D. G. Bailey and D. Lewis, "A checkers playing robot," *Proceedings of the Eleventh Electronics New*, January 2004.

[8] C. Danner and M. Kafafy, "Visual chess recognition," *Stanford University*, January 2015.

[9] T. H. Unger, "Magnus the chess robot," *Norwegian University of Science and Technology*, July 2014.

[10] J. A. A. M. E. S. M. Z. H. L. Eric Brown, Nicholas Rodenberg and H. Jaeger, "Universal robotic gripper based on the jamming of granular material," *Proceedings of the National Academy of Sciences*, vol. 107, pp. 18 809–18 814, November 2010.

[11] S. Russel and P. Norvig, *Artificial Intelligence A Mordern Approach*, 3rd ed. London: Pearson New International Edition, 2014.

[12] J. S. I. A. M. L. A. G. M. L. L. S. D. K. T. G. T. L. K. S. D. H. David Silver, Thomas Hubert, "A general reinforcement learning algorithm thatmasters chess, shogi and go through self-play," *Science*, vol. 362, pp. 1140–1144, December 2018.

[13] N. S. Nise, *Control Systems Engineering*, 7th ed. Hoboken: John Wiley and Sons Inc., 2015.