The University of Hong Kong

2024 – 25

COMP4801

Final Year Project

Interim Report

Autonomous Chinese Checkers Playing Robot

Name	UID
LEUNG Ho Ning	3035801453
SHIU Chun Nam Alex	3035800849

Acknowledgment

Throughout the journey of researching, planning, and implementing this project, we have been fortunate to receive invaluable assistance and guidance from numerous individuals. Their willingness to share their time and experiences has been instrumental in helping me reach this significant milestone—being able to detect the checkerboard and pieces successfully and control the robotic arm smoothly with a self-programmed mobile application. We are deeply grateful for their support and encouragement, and we would like to take this opportunity to express my heartfelt appreciation to everyone who has contributed to the success of this project.

Firstly, we would like to thank our supervisor, Dr. T.W. Chim who has guided us on how to approach and prepare the project. He had inspired us with his ideas on how to present our final desired product during the presentation and had given us a lot of positive feedback every time during our meetings. Thanks to his sharing of his precious time with us, we could find him to discuss on our project weekly.

Next, we would also like to thank the technical staff – Mr. David Lee who has given us lots of instructions on how to build a remote control mobile app to control our robotic arm and how to solve the hardware problem of our robotic arm. He patiently taught us hardware and software techniques to help us transform our ideas from theory to the project. Without his kind help, it would take us much more time to deal with the hardware problem and we may not be able to make such great progress.

Last but not least, we would also like to dedicate our thanks to the Faculty of Engineering and Tam Wing Fan Innovation Wing for providing facilities such as 3D printer and tool kits which facilitated our progress toward finishing the project.

Abstract

Chinese Checkers is a game many of us grew up playing, especially during family gatherings and Lunar New Year celebrations. It holds a special place as a collective memory for many in the Chinese community, symbolizing togetherness and tradition. In this report, we will demonstrate how a robotic arm is implemented and will be combined with AI to form an Autonomous Chinese Checker playing robot product.

Our product consists of a physical robotic arm and a Chinese Checker playing AI. The robotic arm requires active control to be manipulated accurately in moving pieces on the Checkerboard and the AI software takes the jobs of Checkerboard recognition and decision making strategy. By integrating the robotic arm and AI algorithm, the two can be connected together to produce the final product.

The robotic arm applies the fundamental control methods of kinematic forward theory. The arm need to be fine-tuned in order to overcome the positional error during manipulation. What's more, the recognition of Checker board should be accurate since the manipulation depends on it.

In the latest progress of this project, a self-programmed remote control application has been designed to control the robotic arm and put into experiment. Our next step is to add a sucker or grip at the end of the robotic arm in order to perform the pickup and put down mechanism of marbles. For the software part, we will implement the AI algorithm into our final product in the next two months.

Table of Contents

1. Introduction

1.1 Background

Board games have traditionally been social activities requiring physical presence of players. With recent developments in robotics and computer vision, creating interactive gaming experiences between humans and robotic opponents has become increasingly feasible. While some commercial chess-playing robots exist, they remain expensive showcase pieces rarely accessible to the general public.

The rise of artificial intelligence (AI) and robotics has revolutionized traditional strategy games, most notably chess. AI-powered systems like Stockfish (Tong et al., 2023) and AlphaGo (Silver et al., 2016) have demonstrated exceptional capabilities, often surpassing human expertise. From industrial automation to service robots, these technologies continue to enhance human experiences across various domains. However, while games like chess and Go have received significant attention in AI and robotics research, other culturally significant games remain relatively unexplored.

Chinese Checkers (波子棋/跳棋) holds particular cultural significance in Hong Kong and Chinese communities. As a game deeply embedded in traditions like Lunar New Year celebrations and family gatherings, it represents an important part of cultural heritage that deserves preservation and innovation. Despite its rich strategic elements and cultural value, few attempts have been made to develop interactive robotic systems for Chinese Checkers.

1.2 Motivation

Several factors motivated the development of this project. First, there is a clear gap in accessible robotic gaming systems for Chinese Checkers compared to other traditional board games. This presents an opportunity to innovate while preserving cultural heritage. Second, the COVID-19 pandemic highlighted the value of having alternative gaming options when physical gatherings are limited. A robotic opponent could provide an engaging way to practice and enjoy the game independently.

Additionally, the increasing availability of cost-effective robotics components, computer vision systems, and AI frameworks has made it feasible to develop such a system without specialized industrial equipment. This creates an opportunity to demonstrate how modern technology can enhance traditional games while keeping them accessible.

Most importantly, we believe that by doing this, the special cultural significance of Chinese Checkers can be preserved effectively.

1.3 Objectives

The main objectives of this project are:

1. Design and construct a Chinese Checkers playing robot with autonomous capabilities:

- Develop a robotic arm system capable of precise piece manipulation.
- Implement computer vision for accurate board state recognition.
- Create an AI system for strategic gameplay against human players.

2. Create an engaging and accessible user experience:

- Develop an intuitive mobile interface for game control and interaction.
- Ensure smooth and reliable physical piece manipulation.
- Maintain the traditional tactical feel of the game.

3. Build a scalable and modifiable system:

- Design the system to accommodate future enhancements.

- Create a framework that could be adapted for other board games.
- Keep the system cost-effective and reproducible.

1.4 Key Deliverables

This project consists of three main deliverables, or in other words, three phases:

1. Board Recognition System

- Computer vision system using mobile phone camera and computer vision framework.
- Accurate detection of board state and piece positions.
- Real-time position data processing and transfer to control system.

2. Robotic Arm Control

- Precise movement control in X, Y, and Z axes.
- Inverse kinematics calculations for accurate piece manipulation.
- Integration with wireless communication protocol.

3. Game Logic and Control Interface

- Android application for game control.
- Implementation of Artificial Intelligence logic of Chinese Checkers.
- Integration between vision system, AI, and robotic control, a final product

Each component will be developed modularly to allow for independent testing and future enhancements. The system prioritizes reliability and user experience. Currently we have successfully delivered the recognition system and we currently working on the second and third part, which will be explained later.

1.5 Uniqueness of the project

This project stands out for several reasons. Firstly, no significant existing solutions are focusing on Chinese Checkers detection or gameplay automation, unlike other games such as chess. Secondly, it introduces the first autonomous robotic Chinese Checkers-playing system, filling a gap in the fields of AI and robotics. Thirdly, the project emphasizes building a functional, real-world product that offers an interactive gaming experience. Our implementation prioritizes cost-effectiveness through readily available components and 3D-printed parts, making the system more reproducible for educational and research purposes. Finally, it preserves the cultural value of Chinese Checkers by innovating around a game deeply rooted in tradition and collective memory.

1.6 Outline

This report is organized as follows:

Section 2 (Methodologies) outlines our technical approaches for board recognition, robotic control, and game logic implementation, focusing on the completed first phase of vision system development.

Section 3 (Preliminary Results) presents current progress in computer vision implementation and initial hardware assembly, detailing the completed board recognition system and ongoing robotic arm development.

Section 4 (Future Directions) describes upcoming work in Phase 2 (robotic arm control and piece manipulation) and Phase 3 (AI gameplay implementation), along with plans for system integration and refinement.

Finally, Section 5 concludes with a summary of our contributions and insights gained from this project.

2. Methodology

This section details the overview of our project and how we approach to develope an autonomous Chinese Checkers playing system, consist of three main components: board recognition, robotic control, and game strategy. Our methodology emphasizes practical implementation using readily available components.

2.1 Chinese Checker Detection

Board recognition is implemented through traditional computer vision techniques using OpenCV rather than machine learning approaches. The system processes images from the mobile phone camera to detect the board layout, identify piece positions, and map them to a logical game state.

Our implementation employs traditional computer vision techniques using OpenCV rather than machine learning approaches, a decision driven by practical constraints and system requirements. Deep learning solutions, while powerful, require extensive labelled training datasets and significant computational resources for model training. The resources required are typically beyond the scope of a part of the final-year project. Additionally, creating accurate training datasets for Chinese Checkers would be particularly challenging due to the variety of board designs, lighting conditions, and piece variations.

Instead, our OpenCV-based approach, implemented in C++, achieves reliable detection through geometric pattern matching and colour segmentation while remaining computationally efficient. This approach meets our real-time performance requirements on mobile devices and provides the accuracy needed for robotic piece manipulation. Most importantly, our modular system design allows for future enhancements through machine learning when additional resources become available, without requiring a complete system overhaul.

2.2 Robot Arm

Our project employs the RoArm-M2-S, a 4-DOF smart robotic arm featuring a 360° omnidirectional base combined with three flexible joints. The arm's workspace diameter of 1 meter adequately covers our Chinese Checkers board layout, while its joint direct-drive design with 12-bit magnetic encoders achieves 0.088° positioning accuracy essential for precise piece manipulation.

The control system is built around the onboard ESP32 microcontroller, which manages both inverse kinematics calculations and servo motor control. We could develop our control logic using the Arduino IDE, implementing custom movement sequences for piece manipulation through JSON commands, by HTTP request. These commands control individual joint angles and coordinate-based positioning, allowing us to define precise movement patterns for game piece interactions.

We plan to program specific movement sequences optimized for Chinese Checkers, with the integration of RoArm-M2-S:

- Grid positions mapping to the hexagonal board layout
- Vacuum gripper control for secure marble pickup and placement
- Height-optimized movement paths to avoid disturbing other pieces
- Position verification to ensure successful marble transfers

2.3 Artificial Intelligence Algorithm

We plan to implement the game logic using the Minimax algorithm with Alpha-Beta pruning for decision making. The initial implementation will focus on two-player gameplay, as this provides a foundation for more complex multi-player scenarios.

We selected the Minimax algorithm with Alpha-Beta pruning as our strategic decision engine for several key reasons. The algorithm excels at perfect-information, two-player zero-sum games like Chinese Checkers, where each player's actions are fully visible and one player's gain equals the opponent's loss. Additionally, Chinese Checkers' discrete board states and clear winning conditions make it well-suited for tree-based search algorithms.

Minimax works by alternately simulating maximizing (our moves) and minimizing (opponent's moves) turns, assuming optimal play from both sides. In Chinese Checkers, this translates to evaluating:

- Direct paths toward goal positions
- Blocking opponent's optimal trajectories
- Creating or preventing "stepping stone" configurations

However, in initial stage, we plan to focus on the first point for our movement, with end-game optimization.

Alpha-Beta pruning significantly improves search efficiency by eliminating branches that cannot influence the final decision. This optimization is particularly valuable for Chinese Checkers due to its high branching factor - each piece typically has multiple possible moves.

We plan to implement this logic on a server to achieve greater search depth without impacting client performance. The server architecture will support:

- 3-9 ply search depth initially
- Parallel state exploration
- Real-time move calculation via Wi-Fi

This implementation provides a solid foundation for future enhancements, including adaptation to multi-player scenarios and integration of machine learning techniques for position evaluation.

3. Preliminary Results

3.1 Chinese Checker Detection

Chinese Checker detection is the foundation of this project, as it ensures accurate recognition of the board, cells, and marbles and makes the project feasible. The process involves multiple stages of image processing and computer vision techniques to achieve high precision.

3.1.1 Challenges in Detection

Detecting the Chinese Checkerboard and its pieces presents several challenges. Variability in lighting conditions can affect the visibility of the board and marbles. Additionally, reflections and shadows from the marbles introduce noise in the image, complicating the detection process. The circular shape of marbles also requires specific algorithms to ensure proper identification, while the irregular arrangement of cells adds complexity to contour detection. Moreover, the narrow distances between marbles in Chinese Checkers add difficulty to the process.

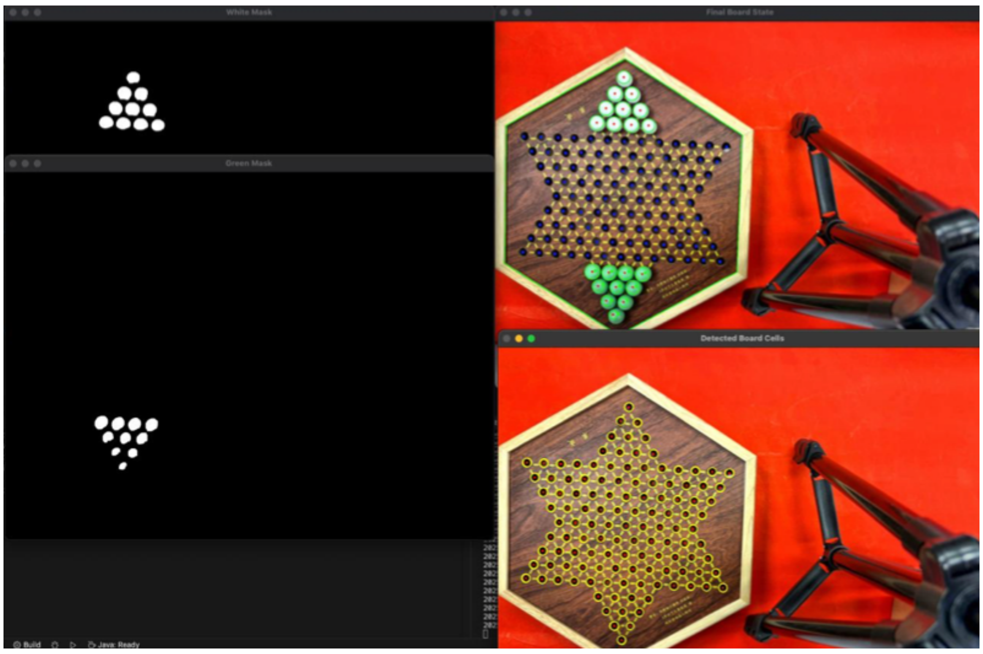

Figure 3.1 A graph showing the difficulty in detection

3.1.2 3D Modeling

To enable accurate mapping and movement calibration for the robotic arm, we have decided to use 3D printing technology to print out our Rhino self-designed checkerboard. We slightly increase the distance between marbles on the board. This ensures that the detection process can be carried out with high accuracy, overcoming the challenges mentioned.

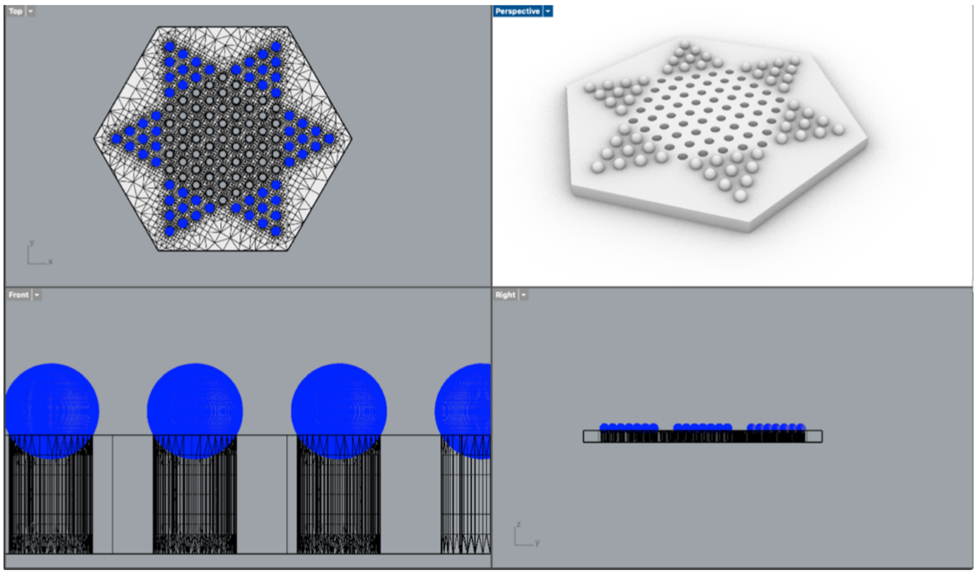

Figure 3.2 Rhino self-designed checkerboard

3.1.3 Computer Vision – Image Processing

Image processing techniques are employed to preprocess the captured images for better detection. The techniques explained below improve the clarity of the image and reduce noise.

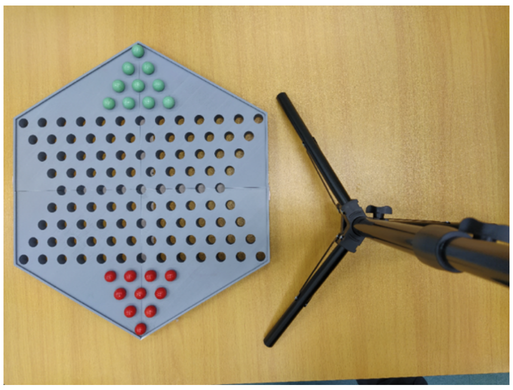

Figure 3.3 Preprocessed photo of our setup

3.1.3.1 Adaptive Resizing

To ensure consistent processing performance and image quality, adaptive sizing is applied as below:

- Calculate the scaling factor based on the maximum dimension (max_dim), which is currently set to 1600 pixels.
- Resize the image using cv2.resize with the interpolation method cv2.INTER_AREA, which is optimized for smoother downscaling.

3.1.3.2 Brightness and Contrast

To effectively enhance brightness and contrast, the image is first converted to grayscale. This simplifies the processing by focusing solely on intensity values, ignoring the color information. This grayscale conversion provides a single intensity channel, which streamlines subsequent analysis.

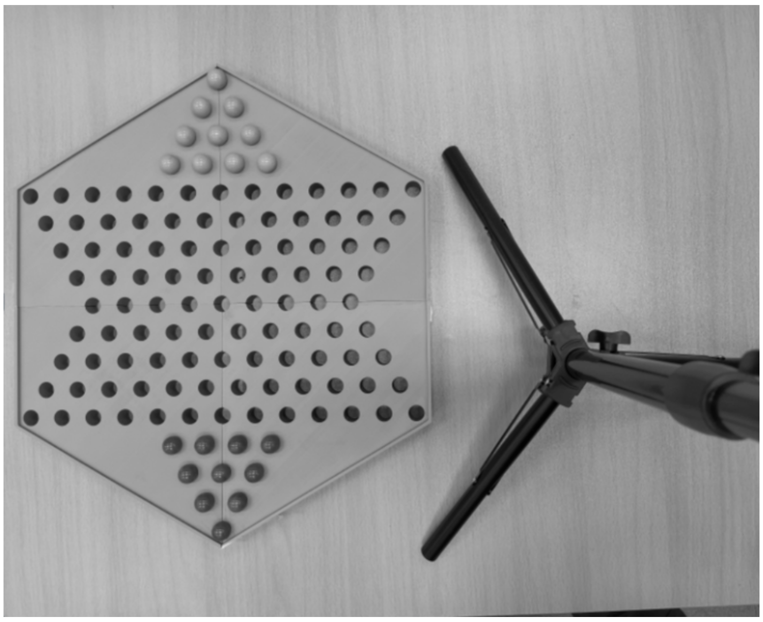

Figure 3.4 Grayscale photo of our setup

Compute the Histogram and Cumulative Histogram

Next, the grayscale histogram is computed using cv2.calcHist, which calculates the frequency of pixel intensities ranging from 0 to 255. For instance, hist[0] represents the number of pixels with an intensity of 0, while hist[1] corresponds to intensity 1, and so forth.

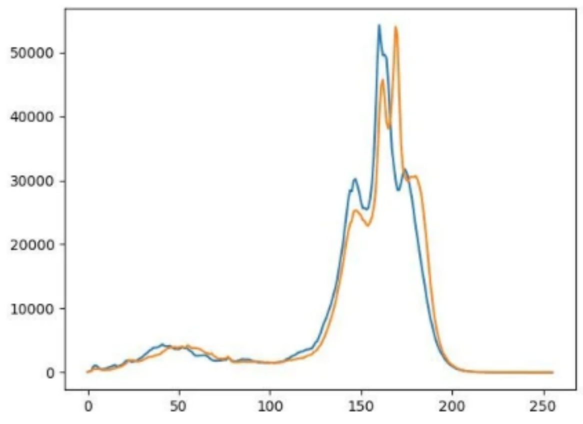

Figure 3.5

Additionally, the cumulative histogram (cum_hist) is derived, representing the running sum of the histogram values. This cumulative measure provides the count of pixels with intensities less than or equal to a given value.

Determine Histogram Clipping Limits

To enhance contrast, histogram clipping is employed, removing outliers that represent extremely dark or bright pixels. Currently, a 3% clipping threshold is applied, which significantly increases the contrast by ignoring extreme values.

Find the Minimum and Maximum Gray Levels After Clipping

Post-clipping, the minimum and maximum grey levels within the valid intensity range are determined.

Compute the Scaling Factor (Alpha) and Bias (Beta)

To adjust the image intensities effectively, two parameters are calculated:

- **Alpha (Contrast Scaling):**

  Alpha is calculated as 255.0 / (maximum_gray - minimum_gray). It is used to calculate the effective range of pixel intensities after clipping. If the difference between maximum_gray and minimum_gray is 0, it indicates no intensity variation (a flat histogram), and alpha is set to 1.0, resulting in no scaling. The goal is to stretch the clipped intensity range to span the full 0-255 scale.

- **Beta (Brightness Adjustment):**

  This parameter shifts the intensity range so that minimum_gray maps to 0. It is calculated as: -minimum_gray * **Alpha**. This ensures that the adjusted intensities are appropriately distributed across the valid range, enhancing the perceived brightness of the image.

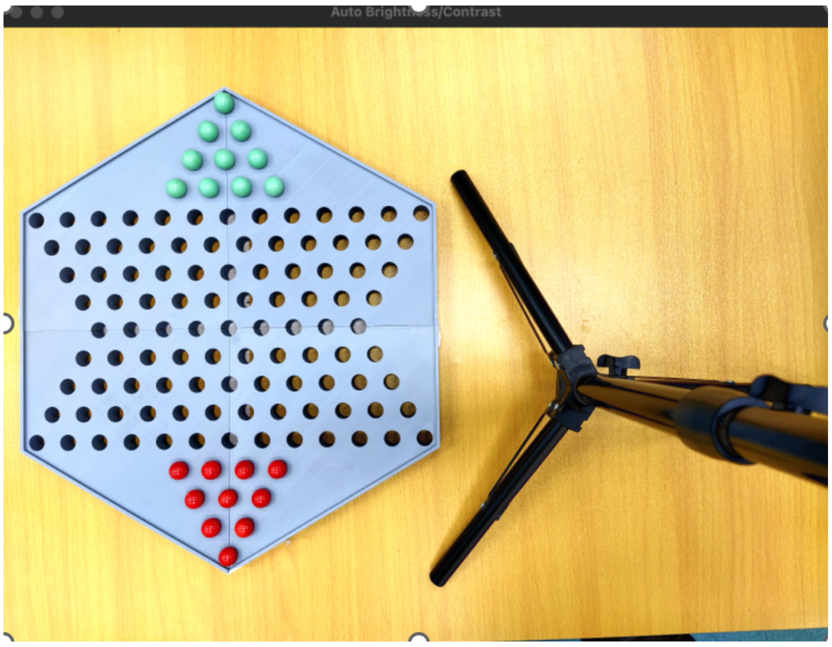

Figure 3.6 Photo of our setup after adjusting brightness and contrast

3.1.3.3 Gaussian Blur

The **Gaussian Blur** operation applies a 2D Gaussian kernel convolution to the image to smooth and reduce high-frequency noise. This process enhances feature detection by simplifying the image, reducing noise, and maintaining the structural integrity of edges. The operation uses the following equation for convolution:

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

Where:

- (x,y) are pixel coordinates relative to kernel center
- σ (sigma) is the standard deviation of the Gaussian distribution, controlling the spread of the blur.
- The kernel size determines the computation window, influencing the degree of smoothing.

In our implementation, OpenCV's cv2.GaussianBlur() function was used with a kernel size of 5×5 and a sigma value set to 0. This configuration lets OpenCV automatically compute the optimal sigma value based on the kernel size.

Gaussian Blur is particularly effective in pre-processing for object detection tasks due to its ability to suppress small-scale noise while preserving important edge details (Gonzalez & Woods, 2018). For instance, this step eliminates minor imperfections that could interfere with circle detection in the Hough Circle Transform and merges small breaks in contours, allowing for continuous boundary identification.

Key Contributions to the Project

1. **Noise Reduction**: By removing small variations in intensity, Gaussian Blur minimizes the impact of lighting inconsistencies and reflections on the marble detection process.

2. **Feature Enhancement**: It simplifies the structure of the image, making circular shapes more prominent and easier to detect.
3. **Robustness**: Gaussian smoothing ensures consistent performance across varying lighting conditions and board designs, enhancing the accuracy of contour detection.

The choice of kernel size (5 × 5) was informed by recommendations from previous literature on computer vision tasks, where smaller kernels are often effective for board game analysis (Russ, 2016). Our use of Gaussian Blur has significantly improved the reliability of our image processing pipeline, particularly for detecting the circular marbles structure unique to Chinese Checkers.

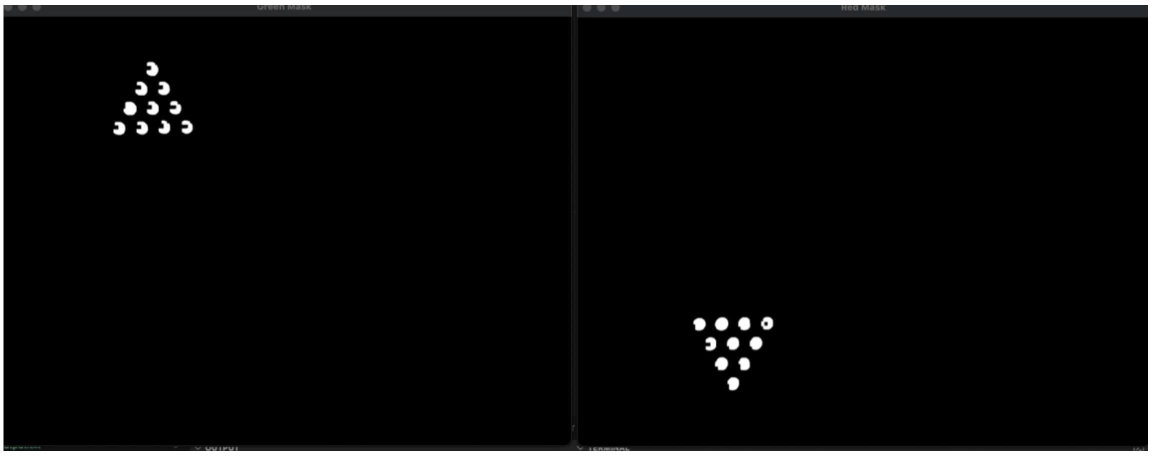
Figure 3.7 Before applying gaussian blur

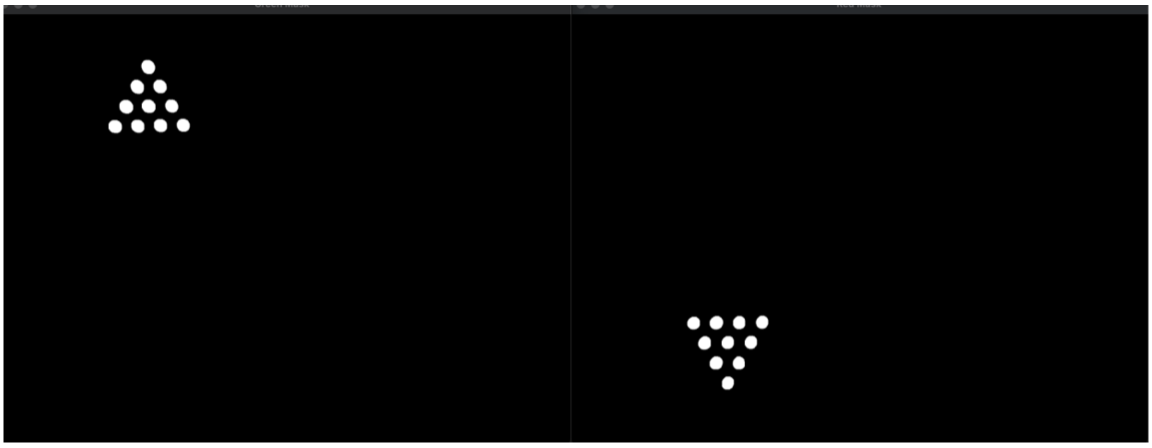
Figure 3.8 After applying gaussian blur

3.1.4 Computer Vision – Board Detection

This sections mainly focus on the techniques we apply to identify its edges and contours, detecting the accurate board area.

3.1.4.1 Adaptive Threshold (Not In Use)

Adaptive thresholding was initially considered for segmenting the Chinese Checkers board and pieces. This technique calculates thresholds for smaller regions of an image, allowing for dynamic adjustments based on local pixel intensities, which can be beneficial in images with varying lighting conditions.

However, in our application, adaptive thresholding yielded inconsistent results. The method's sensitivity to local variations sometimes led to misclassification of board areas and pieces, especially under non-uniform illumination. Consequently, we opted for alternative image processing techniques that provided more reliable segmentation for our specific use case.

3.1.4.2 Canny Edge Detection

In our project, accurately identifying the boundaries of the Chinese Checkers board is essential for effective piece recognition and movement planning. The Canny edge detection algorithm was chosen for its robustness and precision in edge identification. Developed by John F. Canny in 1986, this algorithm is a widely used method for detecting significant edges within an image, particularly in applications requiring high accuracy and low noise interference (Canny, 1986). It is designed to meet three main criteria: low error rate, good localization, and minimal response. The method employs two threshold values—a lower threshold and an upper threshold—to detect edges. According to the OpenCV documentation, the ratio of the upper to lower threshold should generally be between 2:1 and 3:1 for optimal results (OpenCV, n.d.).

The Canny edge detection algorithm operates through a series of steps. First, the image is smoothed using a Gaussian filter to minimize noise, which can otherwise lead to false edge detection. This is achieved by convolving the image with a Gaussian kernel to reduce high-frequency noise. After noise reduction, the algorithm calculates the intensity gradients in both horizontal (G_x) and vertical (G_y)directions. The gradient magnitude (G) and direction (θ) are computed as follows:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Next, non-maximum suppression is applied to thin the edges by retaining only local maxima in the gradient direction, ensuring that the edges are as precise as possible. Following this, double thresholding is performed to classify edge pixels as strong edges (above the high threshold), weak edges (between the thresholds), and non-edges (below the low threshold). Finally, edge tracking by hysteresis connects weak edge pixels to strong edge pixels if they are part of the same edge segment, preserving true edges while discarding noise.

In our implementation, we initially set the lower threshold to 50 and the upper threshold to 150 based on standard practices. However, through iterative testing and refinement, we adjusted the thresholds to 85 (lower) and 255 (upper) for our specific application. This adjustment provided optimal results, ensuring that the edges of the Chinese Checkers board were distinctly and accurately detected. These optimized thresholds enhanced the clarity of edge detection, enabling reliable piece recognition and movement planning critical to the success of our project.

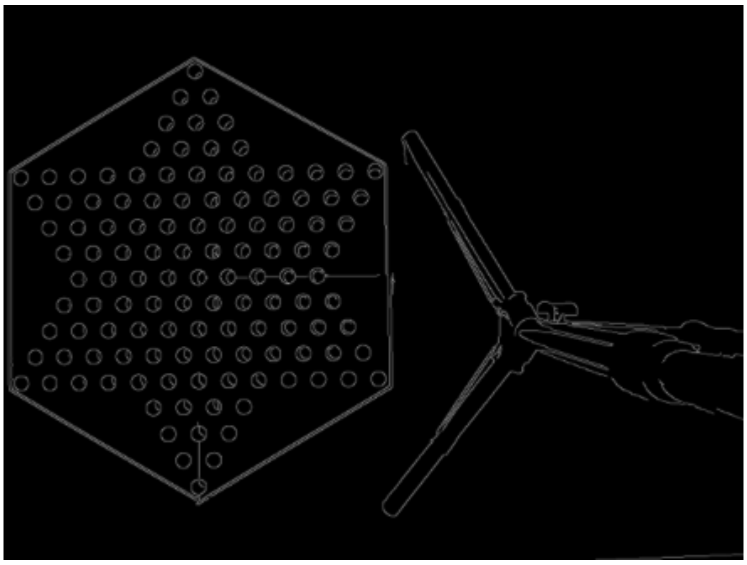

Figure 3.9 Photo using Lower Threshold:50 Upper Threshold:150

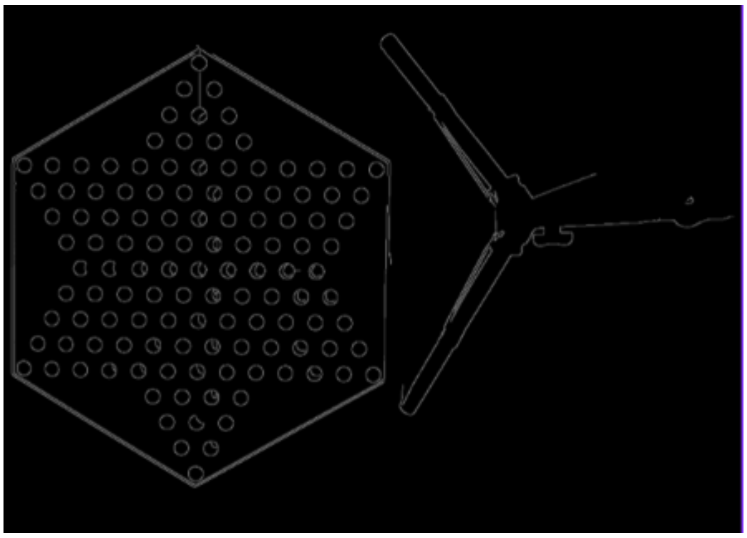

Figure 3.10 Photo using Lower Threshold:80 Upper Threshold:255

3.1.4.3 Morphological Closing

Morphological closing is a vital image processing step used to enhance edge continuity by filling gaps in the edges detected through the Canny edge detection algorithm. This operation is especially important in our project, as it ensures that the boundaries of the Chinese Checkers board form a continuous contour, which is crucial for accurate piece recognition and reliable movement planning.

The morphological closing operation is a combination of dilation followed by erosion. Dilation expands the edges by adding pixels to the boundaries, while erosion contracts the edges by removing pixels from the boundaries. The combination effectively fills small gaps and smooths irregularities in detected edges. The process is mathematically defined as:

$$A \cdot B = (A \oplus B) \ominus B$$

Where:

- A is the binary input image (edges detected by Canny).
- $B$is the structuring element (kernel).
- $\oplus$ represents dilation, and $\ominus$ represents erosion.

The size and shape of the structuring element (B) play a significant role in determining the extent of the closing operation. In our implementation, we used a square kernel with a size of 7×7, as this size was found to be optimal for filling the gaps between the edges of the checkerboard without introducing unwanted artifacts. This choice was informed by empirical testing and the requirements of the project.

By applying morphological closing with a kernel of 7×7, the algorithm ensures that the board's edges are continuous and well-defined. This step significantly improves the accuracy of contour detection and subsequent board state analysis, forming a robust foundation for the overall system. (OpenCV, n.d.)

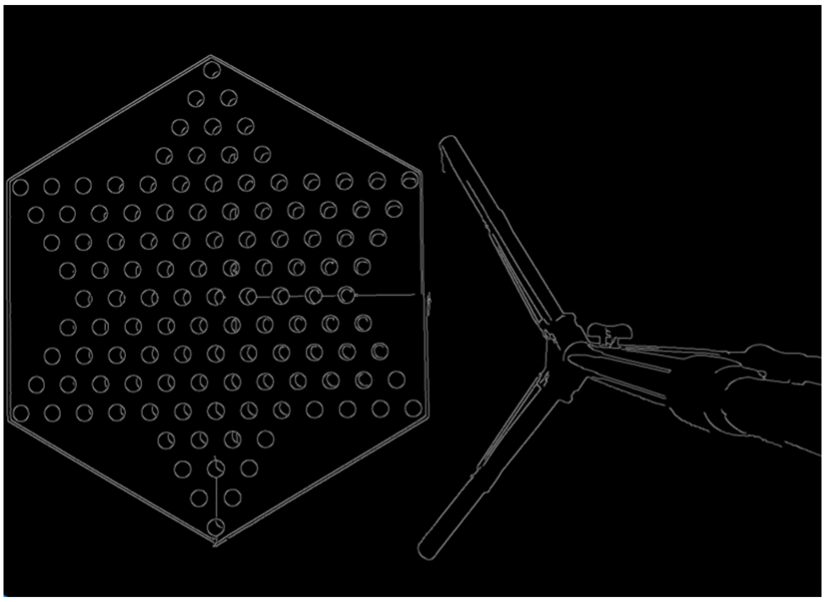

Figure 3.11 Before applying Morphological Closing

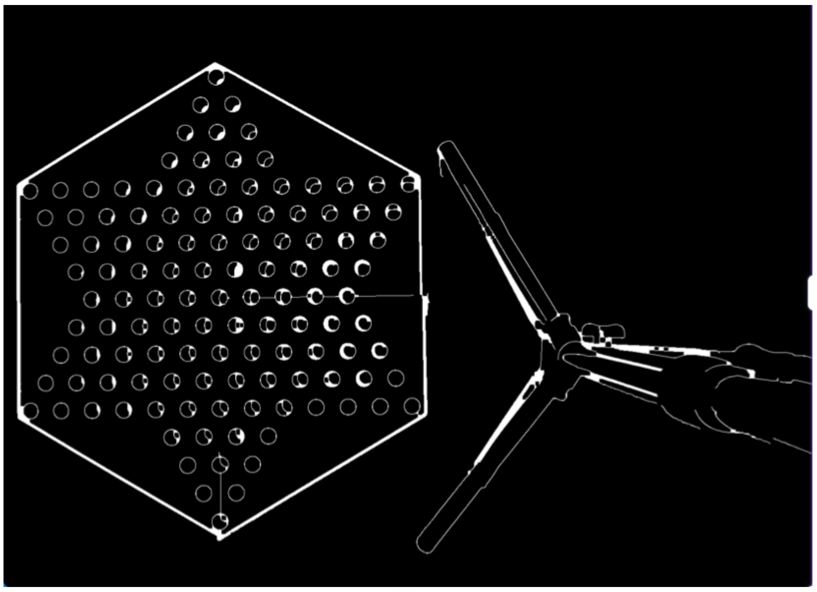

Figure 3.12 After applying Morphological Closing

3.1.4.4 Find Contours

Contour detection is a crucial step in our system, enabling the identification of closed shapes within the processed image. This step is essential for delineating the boundaries of the Chinese Checkers board and other key features. In our implementation, we utilize OpenCV's cv2.findContours() function, which is based on the Suzuki-Abe algorithm (Suzuki & Abe, 1985). The function extracts contours from a binary image, where non-zero pixels represent potential boundaries, allowing the system to identify curves joining continuous points along these edges.

To ensure the focus is on the primary board boundary, we configure the cv2.findContours() function with the RETR_EXTERNAL parameter, which retrieves only the outermost contours, ignoring any internal or nested contours. Additionally, the CHAIN_APPROX_SIMPLE method is employed to reduce redundant points along straight-line segments, retaining only the endpoints and thereby simplifying the contour representation. In our implementation, each contour is represented by 6 to 18 points. This range provides enough tolerance for minor shape irregularities while ensuring the precision necessary for accurate boundary representation.

Once contours are detected, we apply a filtering process to focus only on the most relevant shapes. Contours that occupy less than 5% of the image area are discarded to eliminate noise and insignificant features. Among the remaining contours, only the largest ones are selected for further processing, as they correspond to the board boundary. This approach ensures that the board's edges are clearly defined and free of noise. (OpenCV, n.d.)

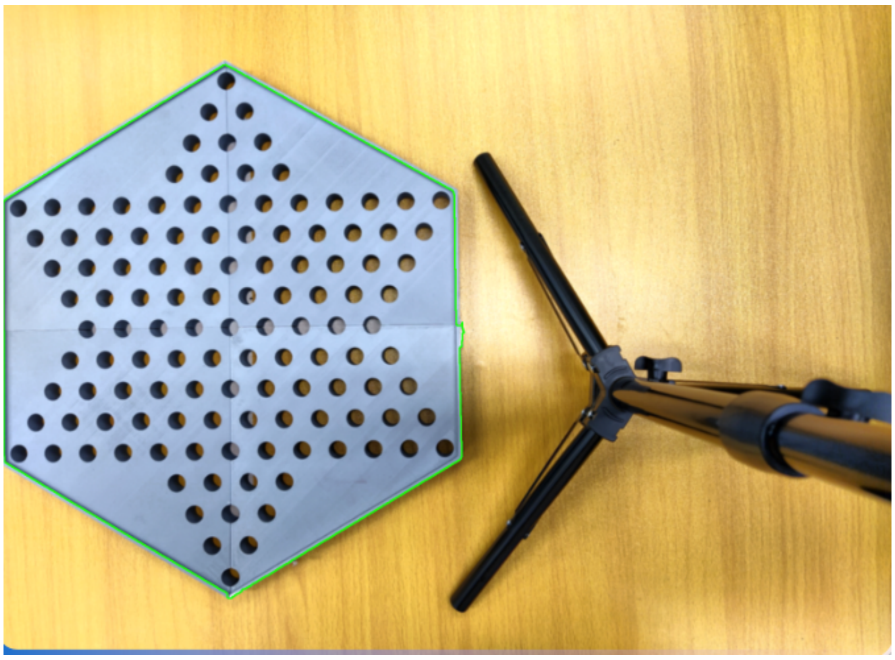

Figure 3.13 After applying Contour detection

3.1.5 Cells Detection

Accurate cell detection is crucial for identifying the positions of marbles on the Chinese Checkers board. This section builds upon the foundational image preprocessing steps previously explained, including **conversion to grayscale**, **application of median blur**, and **brightness enhancement**. These steps simplify the image by reducing color information, removing noise while preserving edges, and improving the visibility of cells and marbles. Together, these techniques create a refined input for the subsequent detection processes. As these theories have already been explained in previous sections, the following part will focus on the theory newly adopted in cells detection.

3.1.5.1 Hough Circle Transform

The **Hough Circle Transform** is an essential technique used to detect circular shapes in our system, facilitating precise identification of cells on the Chinese Checkers board. This method is particularly effective for recognizing circles under challenging conditions, including varying lighting and noise, making it ideal for our application.

Theory of Hough Circle Transform

The Hough Circle Transform is based on the parametric equation of a circle:

$$(x-a)^2+(y-b)^2=r^2$$

Where:

- $(a,b)$are the coordinates of the circle’s center,
- r is the radius.

The algorithm maps each edge pixel from the image space into the parameter space of potential circle centers and radii. Each pixel votes in an accumulator matrix for all circles that could pass through it. Peaks in this accumulator space correspond to the most likely circles. This theoretical approach was first generalized by Ballard (1981) to detect arbitrary shapes, including circles, making it a powerful method for our use case.

The process involves:

1. **Edge Detection**: A binary edge-detected image is created, using the Canny edge detection algorithm.
2. **Voting in Parameter Space**: Each edge pixel votes for circles centered at (a,b)(a, b)(a,b) with a radius rrr. These votes are accumulated in a 3D parameter space (center coordinates a,ba, ba,b and radius rrr).
3. **Peak Detection**: Peaks in the accumulator matrix identify the centers and radii of potential circles.

Implementation in Our System

In our project, we employ OpenCV's cv2.HoughCircles() function to perform the Hough Circle Transform. Key parameters are fine-tuned based on the geometry of the board and the characteristics of the marbles. The parameters are:

- **dp = 1.1**: The inverse ratio of the accumulator resolution to the image resolution. A value close to 1 ensures high precision.
- **minDist = 30**: The minimum distance between detected circle centers. This prevents overlapping detections and ensures that each circle corresponds to a unique cell.
- **param1 = 500**: The upper threshold for the Canny edge detection step, ensuring strong edges are detected while minimizing noise.
- **param2 = 10**: The accumulator threshold for identifying circles. Lower values increase sensitivity but may introduce false positives.
- **minRadius = 15 and maxRadius = 25**: These define the acceptable range of circle radii based on the board's design and cell sizes.

By combining the theoretical strengths of the Hough Circle Transform (Ballard, 1981) with careful parameter tuning and validation (OpenCV, n.d.), our system achieves reliable detection of cells, forming a critical foundation for gameplay interaction and analysis.

3.1.5.2 Board Contour Filtering

Board Contour Filtering ensures that only valid circles detected by the Hough Circle Transform are retained, specifically those corresponding to the cells within the board's boundaries. This step uses OpenCV's *cv2.pointPolygonTest()* function to verify whether the center of each detected circle lies inside, on, or outside the contour of the board. The process eliminates false positives caused by noise or reflections outside the playable area.

The **cv2.pointPolygonTest()** function calculates the shortest signed distance from a point $P(x, y)$ (e.g., the center of a detected circle) to the edges of a given contour C. The contour C is defined as a closed sequence of connected line segments.

Mathematically, the function evaluates:

$$d = \min_{E \in C} \parallel P - E \parallel$$

where:

- $P(x, y)$ is the point being tested.
- $E(x_e, y_e)$ represents points along the edges of the contour CCC.
- $\parallel P - E \parallel = \sqrt{(x - x_e)^2 + (y - y_e)^2}$is the Euclidean distance between the point PPP and a point EEE on the contour.

The **sign of** d indicates the position of the point relative to the contour:

- $d > 0$: The point lies **inside** the contour.
- $d = 0$: The point lies **on** the contour.
- $d < 0$: The point lies **outside** the contour.

(OpenCV, n.d.).

Theory

The **Ray-Casting Algorithm** is used in cv2.pointPolygonTest() to determine whether a point is inside or outside a polygon:

1. A ray is cast from the point $P(x, y)$ to infinity in any direction.
2. The number of times the ray intersects the edges of the contour $C$is counted.
3. If the number of intersections is odd, the point is inside the contour ($d > 0$:); if even, the point is outside ($d < 0$).

In addition to ray-casting, the function calculates the minimum distance from the point to any segment of the contour, providing a signed distance d.

3.1.6 Marble Detection

Marble Detection is a critical step in distinguishing marbles from the board and accurately identifying their positions. The process combines color segmentation and shape analysis to detect marbles of different colors, ensuring only valid marbles within the board are retained. Now we mainly detect for green and red marbles first, which will extend to other colours in future enhancements.

Implementation in Our System

Below are details on how we implement our marble detection algorithm. For clarity, only the parts not previously discussed will be elaborated, specifically the circularity check. Other theories mentioned earlier will be briefly referenced for continuity.

1. **Color Segmentation**: Using predefined HSV thresholds, the system creates binary masks for each marble color:
    - **Green Marbles**: A mask is created using HSV bounds for green, defined as GREEN_LOWER and GREEN_UPPER.
    - **Red Marbles**: Two masks are created for red marbles, covering both ends of the red hue spectrum (RED_LOWER1, RED_UPPER1, RED_LOWER2, and RED_UPPER2). These masks are combined using a bitwise OR operation.

2. **Morphological Cleaning**: To improve mask quality and eliminate noise, morphological operations are applied:
    - **Opening**: Removes small artifacts by eroding and then dilating the mask.
    - **Closing**: Fills small gaps in detected regions by dilating and then eroding the mask.

    The cleaned masks are used to isolate marble regions effectively.

3. **Contour Analysis**: The cv2.findContours() function identifies contours in the masks. For each contour:
    - A **minimum enclosing circle** is fitted using cv2.minEnclosingCircle(), providing the center coordinates (x, y) and radius.
    - The radius is validated to ensure it falls within predefined bounds (MARBLE_HOUGH_MIN_RADIUS = 10, MARBLE_HOUGH_MAX_RADIUS = 30).

4. **Circularity Check**: The circularity check is a critical step that ensures the detected regions resemble marbles, which are expected to be near-perfect circles. This check calculates the circularity of each contour using the formula:

$$\text{Circularity} = \frac{4\pi \cdot \text{Area}}{\text{Perimeter}^2}$$

   Here, Area refers to the area enclosed by the contour, and Perimeter is the total length of the contour boundary. Circularity values range from 0 for irregular shapes to 1 for a perfect circle.

   The rationale for this check lies in its ability to filter out irrelevant or irregular shapes that might otherwise be misclassified as marbles. Since marbles are expected to be nearly circular, this method helps the system distinguish between valid marbles and artifacts caused by noise, overlapping objects, or irregular shapes. A threshold of 0.6 is used, meaning that only contours with a circularity value of 0.6 or higher are considered sufficiently circular to qualify as marbles.

5. **Board Validation**: Using cv2.pointPolygonTest(), the system checks whether the center of each detected marble lies inside the board contour:
    - If the result $d \geq 0$ (inside or on the board), the marble is retained.
    - If $d < 0$ (outside the board), the marble is discarded.

3.1.7 Mapping Logic / Board State

The detected positions of the marbles are used to determine the current state of the board, which is essential for gameplay analysis and strategic decisions. The board's state is encoded into a structured array format that maps each cell to its corresponding marble.

Board Layout

The Chinese Checkers board layout is predefined as a 17-row structure, with each row having a specific number of cells. Currently it has a strcutre as follows:

```
board_layout = [
    [None],               # row 0 has space for 1 cell
    [None, None],         # row 1 has space for 2 cells
    [None, None, None],   # row 2 has space for 3 cells
    [None, None, None, None],    # row 3 has space for 4 cells
    [None, None, None, None, None, None, None, None, None, None, None, None, None],
    [None, None, None, None, None, None, None, None, None, None, None, None
    [None, None, None, None, None, None, None, None, None, None, None],
    [None, None, None, None, None, None, None, None, None, None],
    [None, None, None, None, None, None, None, None, None],
    [None, None, None, None, None, None, None, None, None, None],
    [None, None, None, None, None, None, None, None, None, None, None],
    [None, None, None, None, None, None, None, None, None, None, None, None],
    [None, None, None, None, None, None, None, None, None, None, None, None, None],
    [None, None, None, None],   # row 13 has space for 4 cells
    [None, None, None],         # row 14 has space for 3 cells
    [None, None],               # row 15 has space for 2 cells
    [None],                     # row 16 has space for 1 cell
]
```

This layout is represented as a 2D array, where `None` placeholders indicate unoccupied spots. The layout serves as the reference for assigning cell coordinates and mapping marble positions.

However, in out Artificial Intelligence logic, we adapted a following graph structure in achive of a path searching.

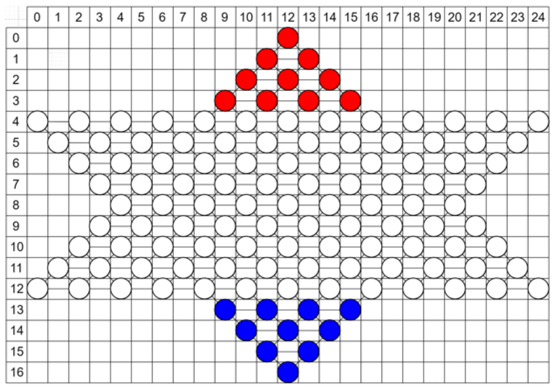

Figure 3.14 Graph for path searching

Groupin Cells into Rows

Detected cells are first grouped into rows based on their y-coordinates. The logic for grouping is implemented in the group_cells_by_row() function, which follows these steps:

1. **Sort by y and x**: Detected cells are sorted by ascending y-coordinates to group them into rows. For cells with the same y-coordinate, x-coordinates are used as a tiebreaker.
2. **Row Thresholding**: A row is defined by grouping cells whose y-coordinates differ by no more than a predefined threshold (row_threshold=15). When the y-difference exceeds this threshold, a new row is started.
3. **Sort by x within Rows**: Each row is further sorted by x-coordinates to maintain the natural left-to-right order of cells.
4. **Flatten Rows**: The grouped cells are combined into a single list, organized row-by-row.

This grouping logic ensures that cells are accurately aligned with the board's layout, even if slight detection deviations occur. The flexibility of the row_threshold parameter allows for adaptation to variations in board geometry.

Assigning Cells to the Board Layout

The grouped cells are then mapped to the predefined Chinese Checkers board layout using the assign_cells_to_layout() function:

1. **Verify Cell Count**: The number of detected cells is compared with the total spots required by the layout. Warnings are logged if fewer cells are detected, indicating a potential issue in earlier detection stages.
2. **Row-by-Row Mapping**: Each row of grouped cells is assigned to the corresponding row in the board layout. Cells are added sequentially, ensuring consistency with the predefined structure.

Mapping Marbles to Cells

The assign_marbles_to_cells() function is used to map each detected marble to the nearest available cell within a defined threshold. The process involves:

1. **Calculating Distances**: For each marble, the Euclidean distance (ddd) to all cell centers is computed:

$$d = \sqrt{(x_{\text{marble}} - x_{\text{cell}})^2 + (y_{\text{marble}} - y_{\text{cell}})^2}$$

   where $(x_{\text{marble}}, y_{\text{marble}})$ are the marble's coordinates and $(x_{\text{cell}}, y_{\text{cell}})$ are the cell's coordinates.

2. **Assigning to the Nearest Cell**: Marbles are assigned to the nearest cell within a threshold distance. The threshold is dynamically adjusted based on the marble's radius (base_threshold + radius) to account for variations in marble size.

3. **Avoiding Overlaps**: Once a marble is assigned to a cell, the cell is marked as occupied to prevent multiple assignments.

This step produces a dictionary that maps each cell to its assigned marble color (green, red, or None).

Error Reduction and Heuristics

To further reduce errors and handle edge cases where marbles may not be detected accurately, a heuristic can be added: if a specific color (e.g., red or green) is observed on a cell, that cell is automatically marked as occupied by a marble of that color. This fallback ensures that the board state remains consistent even in the presence of detection inaccuracy.

3.1.8 Communication Architecture

The communication architecture of the detection system integrates the Android application with a Flask-based server currently, facilitating the transmission of board states and image data for processing and visualization. The architecture operates on a client-server model, as depicted in the diagram, and incorporates the following components:

Android Application (Client-Side):

- The Android app captures images of the board using its camera functionality, implemented with Android's CameraX API.
- These images are converted to a Base64 format and transmitted to the server via HTTP POST requests. Two endpoints are supported:
    - /upload_empty_board: Sends an image of the empty board for initial setup and calibration.
    - /detect_current_state: Sends an image of the board with marbles to determine its current state.
- Upon receiving responses from the server, the app visualizes the detected board state or displays errors as needed.

Flask Server (Server-Side):

- The server, currently hosted on a Mac system, listens on port 5001 for incoming requests.
- Images sent by the client are processed using OpenCV algorithms to extract the state of the board.
- The server maintains debug files and processed images for troubleshooting and refinement of detection accuracy.

OpenCV Processing:

- The server leverages OpenCV for tasks like image preprocessing, contour detection, and color segmentation. The outputs include:
    - **Board State JSON:** Encodes the board layout with cell states (empty or occupied, along with the color of marbles).

- **Debug Files:** Includes annotated images and logs for refining the processing pipeline.

Data Flow:

- The Android application transmits Base64-encoded images as payloads in HTTP POST requests.
- The Flask server processes these images and responds with JSON representations of the board state, detailing each cell's status.
- The app parses this JSON and updates its UI to reflect the detected state.

Scalability and Future Enhancements:

- The architecture is modular, supporting running on cloud server with future extensions like additional marble colors, real-time updates, or alternative client applications.

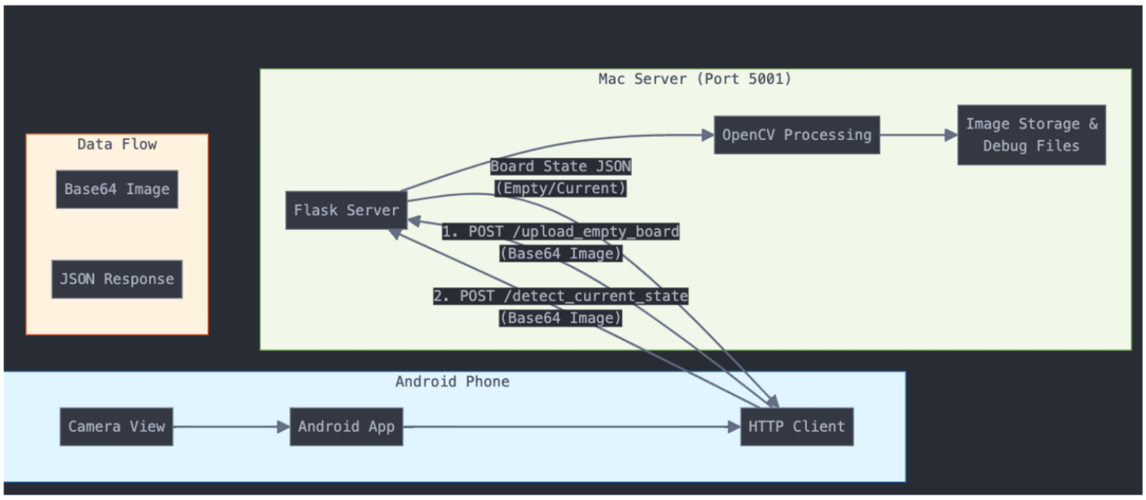
Mac Server (Port 5001)
Data Flow
Base64 Image
JSON Response
OpenCV Processing
Image Storage & Debug Files
Board State JSON (Empty/Current)
Flask Server
1. POST /upload_empty_board (Base64 Image)
2. POST /detect_current_state (Base64 Image)
Android Phone
Camera View
Android App
HTTP Client

Figure 3.15 Flow chart of our System Communication Architecture

3.2. Robot Arm

The RoArm-M2-S is a 4-degree-of-freedom (4-DOF) smart robotic arm designed for innovative applications. It features a lightweight structure, advanced control systems, and customizable end-of-arm tooling (EoAT), making it suitable for various tasks requiring precision and flexibility.

3.2.1 Inverse Kinematics Approach

For a 4-DOF robotic arm like the RoArm-M2-S, inverse kinematics can be approached analytically by solving geometric relationships between the arm's links and joints. The goal is to determine the joint angles (θ_1, θ_2, θ_3, θ_4) that achieve a specified end-effector position (x, y, z) and orientation (φ).

1. **Base Joint (θ_1):**

The base joint angle is determined by the projection of the end-effector position onto the XY-plane:

$$\theta_1 = \arctan\ 2(y, x)$$

2. **Wrist Center Position:**

Calculate the position of the wrist center (the point where the wrist joint connects to the end-effector) by subtracting the end-effector's offset due to its orientation:

$$x_w = x - a_4 \cos\ \theta_1 \cos\ \phi$$

$$y_w = y - a_4 \sin \theta_1 \cos \phi$$

$$z_w = z - a_4 \sin \phi$$

Where:

- x_w, y_w, z_wCoordinates of the wrist center.
- a_4Length of the fourth link (distance from the wrist joint to the end-effector).

- ϕ: Desired orientation angle of the end-effector.

3. **Shoulder and Elbow Joints (θ_2 and θ_3):**

Define intermediate variables:

$$r = \sqrt{x_w^2 + y_w^2}$$

$$s = z_w - d_1$$

$$D = \frac{r^2 + s^2 - a_2^2 - a_3^2}{2a_2a_3}$$

Where:

- r: Horizontal distance from the base to the wrist center.
- s: Vertical distance from the base to the wrist center.
- d_1: Offset along the base joint's Z-axis (distance from the base to the shoulder joint).
- a_2: Length of the second link (distance between the shoulder and elbow joints).
- a_3: Length of the third link (distance between the elbow and wrist joints).

Then, the elbow joint angle is:

$$\theta_3 = \arctan\ 2(\sqrt{1 - D^2}, D)$$

And the shoulder joint angle is:

$$\theta_2 = \arctan 2(s, r) - \arctan 2(a_3 \sin\theta_3\,, a_2 + a_3 \cos\theta_3)$$

4. **Wrist Joint (θ_4):**

The wrist joint angle is calculated to achieve the desired end-effector orientation:

$$\theta_4 = \phi - (\theta_2 + \theta_3)$$

The RoArm-M2-S utilizes an inverse kinematics function to compute the necessary joint angles for a given end-effector position. This allows the robotic arm to accurately reach target points by calculating the rotation angle of each joint.

3.2.2 Control System

The RoArm-M2-S robotic arm is equipped with an ESP32-WROOM-32 microcontroller as its main control unit. This powerful MCU supports various control interfaces and wireless communication protocols, enabling flexible and efficient control of the robotic arm.

ESP32 Microcontroller

The ESP32 is a dual-core microcontroller with integrated Wi-Fi and Bluetooth capabilities. In the RoArm-M2-S, it manages the processing of inverse kinematics calculations, controls the serial bus servos, and handles communication with external devices. The ESP32's features include:

- **Wireless Communication:** Supports 2.4GHz Wi-Fi and Bluetooth, allowing for wireless control and programming.
- **Multiple I/O Ports:** Provides various input/output ports for connecting sensors, servos, and other peripherals.
- **High Processing Speed:** Operates at a clock speed of up to 240MHz, ensuring quick and responsive control operations.

Control Interfaces

The RoArm-M2-S offers multiple control interfaces to accommodate different user preferences and application requirements:

- **Serial Communication:** For advanced control, users can connect to the ESP32 via USB or UART to send JSON commands directly. This method is suitable for integrating the robotic arm into custom applications or for development purposes.
- **ESP-NOW Protocol:** The ESP32 supports ESP-NOW, a low-latency, peer-to-peer wireless communication protocol. This feature allows multiple robotic arms to communicate and coordinate with each other without the need for a Wi-Fi network infrastructure.

Vacuum Gripper Integration

To enhance the versatility of the RoArm-M2-S, a vacuum gripper is planned to be integrated as the end-effector. The robotic arm comes with an expansion plate that supports customizing the End of Arm Tooling (EoAT), allowing for the attachment of various tools, including vacuum grippers. The integration process involves connecting the vacuum gripper to the appropriate I/O ports on the ESP32 controller, which the control could be programmed using the Arduino Integrated Development Environment.

3.3 Artificial Intelligence

The artificial intelligence component of our Chinese Checkers system employs game theory principles through the Minimax algorithm with Alpha-Beta pruning optimization. This section outlines the theoretical foundations of our approach and the mathematical framework underlying our implementation.

3.3.1 Game State Representation

Chinese Checkers can be represented as a finite, deterministic, perfect-information game. The game state S is defined by the tuple:

$$S = (P, B, T)$$

Where:

- P represents the current player ($P \in \{1, 2\}$)
- B denotes the board state matrix
- T indicates the turn number

The board state B is represented as a 17×17 matrix, where each cell bij contains either 0 (empty), 1 (player 1's piece), or 2 (player 2's piece). This representation allows for efficient state evaluation and move generation.

3.3.2 Minimax Algorithm with Alpha-Beta Pruning

The Minimax algorithm, enhanced with Alpha-Beta pruning, serves as the core of our decision-making process in the Chinese Checkers AI system. This approach enables efficient exploration of potential game states by systematically evaluating possible moves and counter-moves, thereby identifying optimal strategies.

Minimax Algorithm with Alpha-Beta Pruning

The Minimax algorithm operates on the principle of minimizing the possible loss in a worst-case scenario. In the context of a two-player game like Chinese Checkers, one player aims to maximize their advantage (MAX), while the opposing player seeks to minimize it (MIN). The algorithm recursively evaluates the game tree by considering all possible moves, assuming that both players play optimally.

For a given game state S, the value function $V(s)$ is defined recursively as follows:

If s is a terminal state (i.e., the game has ended), then:

$$V(s) = \text{eval}(s)$$

Here, $\text{eval}(s)$is the evaluation function that assigns a numerical value to the terminal state based on the outcome (win, lose, or draw).

If s is not a terminal state:

- If the current player is MAX:

$$V(s) = \max_{s' \in M(s)} V(s')$$

- If the current player is MIN:

$$V(s) = \min_{s' \in M(s)} V(s')$$

In these expressions:

- $M(s)$denotes the set of possible moves from state sss.
- s'' represents a successor state resulting from a move in $M(s)$
- $V(s')$is the value of the successor state, determined recursively.

Alpha-Beta Pruning

Alpha-Beta pruning is an optimization technique applied to the Minimax algorithm to reduce the number of nodes evaluated in the game tree, thereby enhancing computational efficiency without affecting the outcome. It introduces two parameters:

- **Alpha (α)**: The best (highest) value that the MAX player can guarantee so far.
- **Beta (β)**: The best (lowest) value that the MIN player can guarantee so far.

During the recursive evaluation of the game tree:

- At MAX nodes, if $V(s)$becomes greater than or equal to β\betaβ, further exploration of that node's descendants is halted (pruned), as the MIN player will avoid this path.
- At MIN nodes, if $V(s)$becomes less than or equal to α\alphaα, further exploration is similarly pruned, as the MAX player will avoid this path.

By maintaining and updating α and β during the search, the algorithm effectively eliminates branches that cannot influence the final decision, thereby reducing the effective branching factor and improving search efficiency.

3.3.3 State Evaluation Functions

In our Chinese Checkers AI system, we employ two distinct evaluation functions to assess the desirability of game states, each capturing different strategic aspects of gameplay. We currently have two design of the evaluation function.

Evaluation Function 1: Distance-Based Progression

The first evaluation function, $E_1(s)$, emphasizes the advancement of pieces toward their target destination. It is defined as:

$$E_1(s) = \sum(w_i \times d_i) + k \times t_i$$

Where:

- w_i: Weight assigned to piece i
- d_i: Normalized distance of piece iii to its goal
- k: Constant bonus multiplier, currently 5
- t_i: Indicator function, 1 if piece $i$is in the target triangle, 0 otherwise

This function rewards pieces for progressing toward their goal and provides additional bonuses for occupying the target area, encouraging efficient movement across the board.

Evaluation Function 2: Board Control

The second evaluation function, $E_2(s)$focuses on controlling key areas of the board, both vertically and horizontally. It is expressed as:

$$E_2(s) = \sum(r_i \times h_i) + c \times g_i$$

Where:

- r_i: Row position weight for piece iii
- h_i: Horizontal position factor for piece iii
- c: Constant associated with goal zone control, currently 50
- g_i: Indicator function, 1 if piece iii occupies a strategic goal zone position, 0 otherwise

In our current implementation, we have designed two distinct evaluation functions, each capturing different strategic aspects of Chinese Checkers. At this stage, we are evaluating their effectiveness to determine which will serve as the final evaluation function. Alternatively, we may provide users with the option to select their preferred evaluation function during gameplay.

3.3.4 Move Generation and Search Space

Efficient move generation is crucial for the performance of our Chinese Checkers AI, given the game's complex branching factors. Our system employs a graph traversal approach to generate all valid moves for a given game state.

For a piece located at position p, the set of valid moves $V(p)$is defined as:

$$V(p) = D(p) \cup J(p)$$

Where:

- $D(p)$: Direct moves to adjacent empty spaces
- $J(p)$: Jump sequences over other pieces

Direct Moves ($\boldsymbol{D(p)}$)

Direct moves involve moving a piece to any immediately adjacent empty space. This is straightforward and involves checking the neighboring positions around p for availability.

Jump Sequences ($\boldsymbol{J(p)}$)

Jump moves allow a piece to hop over an adjacent piece into an empty space directly opposite. Multiple consecutive jumps are permitted within a single move, enabling significant advancement across the board. To generate all possible jump sequences, our system performs a depth-first search (DFS) from the starting position ppp. This recursive approach explores all potential jump paths, ensuring that each valid sequence is considered.

Our implementation adapts these theoretical foundations to create a practical game-playing system, with specific parameter values and optimization techniques determined through empirical testing. Future enhancements will focus on improving evaluation function accuracy and search efficiency through pattern recognition and endgame databases.

4. Future Work

4.1 AI Strategy for Chinese Checkers

To enhance our AI's performance in Chinese Checkers, we are refining our two evaluation functions to better assess game states and inform decision-making. This involves fine-tuning the parameters and weights within each function to more accurately reflect strategic priorities, such as piece advancement and board control. By conducting extensive testing and analysis, we aim to determine the most effective evaluation criteria, thereby improving the AI's move selection process.

Additionally, we plan to deploy our AI system online, leveraging cloud-based infrastructure to achieve deeper search depths during gameplay. Cloud computing offers scalable resources that can handle the intensive computations required for exploring extensive game trees. This approach will increase the search depth for analysis of potential moves, leading to improved strategic planning and overall performance.

4.2 Enhancement of Robotic Arm

Improving the robotic arm's precision and efficiency is another critical focus. Currently, the movement calibration is based on mapping the checkerboard into an array and calculating exact X, Y, Z coordinates for each cell. Future efforts will aim to enhance this system by implementing more advanced calibration methods to further refine the accuracy of pick-and-place operations. Additionally, introducing a vacuum sucker at the end of the robotic arm, will enable the arm to perform pick-up and place movement. Enhancements in other parts of the hardware, such as upgrading the arm's motor capabilities, will also ensure smoother handling of the checker pieces. Our final goal would be to introduce a robotic arm that can reliably and precisely interact with every cell on the checkerboard.

4.3 Integration of Application, Artificial Intelligence, and Arm

Our project's primary objective is to seamlessly integrate the AI algorithm, robotic arm, and mobile application into a cohesive product that offers users an intuitive and efficient setup process. By developing a well-structured communication architecture, we aim to synchronize the AI's strategic decisions with the robotic arm's movements, all managed through a user-friendly mobile interface. This design ensures that users can quickly and easily engage with the system, enjoying an autonomous and interactive Chinese Checkers-playing experience without complex configurations.

To achieve this, we are focusing on creating a plug-and-play system that minimizes setup time and technical barriers for users. The mobile application will serve as the central hub, facilitating seamless communication between the AI and the robotic arm. By prioritizing ease of use and quick installation, we aim to make our product accessible to a broad audience, allowing everyone to enjoy the innovative blend of artificial intelligence and robotics in a traditional game setting.

4.4 Schedule and Milestone

Date	Milestone
Sep 2024	Deliverables of Phase 1 (Done) • Detailed project plan • Project web page
Oct-Nov 2024	• Modify the robotic arm • Make a Demo version of mobile app
Dec 2024	Deliverables of Phase 2 (Done) • Further develop app so that it can control the robotic arm to execute the instructions • Implement checkboard and pieces recognition function into the app
Jan-Feb 2025	Deliverables of Phase 3 (Pending) • Apply Minimax algorithm and Alpha Beta Pruning to the AI application
Apr 2025	Final presentation

5. Conclusion

The successful development of an autonomous Chinese Checkers playing robot represents a significant achievement in merging traditional gaming with modern technology. Through the integration of computer vision, robotics, and artificial intelligence, this project has created a system that not only preserves the cultural significance of Chinese Checkers but also advances the field of interactive gaming technology. The modular approach to system development demonstrates the feasibility of creating accessible autonomous gaming systems.+

The project has currently achieved reliable board state detection through computer vision and established basic robotic control via a mobile interface. With these foundations in place, development now focuses on enhancing the AI's strategic capabilities and implementing precise piece manipulation through a vacuum gripper system. When completed, this cost-effective and reproducible system will serve as a blueprint for autonomous gaming platforms, demonstrating how technological innovation can enhance traditional games while maintaining their cultural essence.

References List

Ballard, D. H. (1981). Generalizing the Hough Transform to detect arbitrary shapes. *Pattern Recognition, 13*(2), 111–122.

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6), 679–698.

Gonzalez, R. C., & Woods, R. E. (2018). *Digital image processing* (4th ed.). Pearson.

OpenCV. (n.d.). Canny Edge Detection. Retrieved from https://docs.opencv.org/4.x/da/d5c/tutorial_canny_detector.html

OpenCV. (n.d.). Contours hierarchy and retrieval modes. Retrieved from https://docs.opencv.org/4.x/d9/d8b/tutorial_py_contours_hierarchy.html

OpenCV. (n.d.). Hough Circle Transform. Retrieved from https://docs.opencv.org/4.x/da/d53/tutorial_py_houghcircles.html.s

OpenCV. (n.d.). Morphological transformations. Retrieved from https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

OpenCV. (n.d.). *Point Polygon Test*. Retrieved from https://docs.opencv.org/3.4/dc/d48/tutorial_point_polygon_test.html.

Russ, J. C., & Neal, F. B. (2016). *The image processing handbook* (7th ed.). CRC Press.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Hassabis, D. (2017). Mastering the game of Go without human knowledge. Nature, 550(7676), 354-359.

Suzuki, S., & Abe, K. (1985). Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing, 30(1), 32–46.

Tong, A., Perez, J., & Stockfish Team. (2023). Stockfish: A strong open source chess engine. arXiv preprint arXiv:2023.xxxxx.