The University of Hong Kong

2024 – 25

COMP4801

Final Year Project

Final Report

Autonomous Chinese Checkers Playing Robot

Name	**UID**
LEUNG Ho Ning	3035801453
SHIU Chun Nam Alex	**3035800849**

Acknowledgment

This project would not have been possible without the invaluable guidance, support, and encouragement of many individuals. Their expertise and generosity of time enabled us to develop robust checkerboard detection, precise piece recognition, and smooth robotic-arm control via our custom mobile application.

First and foremost, we extend our deepest gratitude to our supervisor, Dr. T. W. Chim. His insightful suggestions on project methodology and presentation design, together with his consistently positive feedback during our meetings, were instrumental in shaping both our technical approach and our final deliverable.

We are also profoundly grateful to Mr. David Lee from the technical staff, whose patience and hands-on instruction in mobile-app integration and hardware troubleshooting turned theoretical ideas into a working robotic system. His mentorship dramatically accelerated our progress and helped us overcome complex mechanical and software challenges.

Finally, we wish to acknowledge the Faculty of Engineering and the Tam Wing Fan Innovation Wing for providing access to 3D printing facilities, tool kits, and workspace. Their support created the perfect environment for innovation and allowed us to bring this autonomous Chinese Checkers robot to fruition.

Abstract

Chinese Checkers, a game deeply rooted in cultural traditions and familial bonds, symbolizes unity and heritage within Chinese communities. This project reimagines this classic game through the lens of modern robotics and artificial intelligence (AI), creating an autonomous system that bridges tradition and innovation. The developed product integrates a 4-DOF robotic arm with a vision-driven AI engine, enabling human-robot gameplay with precision and strategic depth.

The robotic arm, powered by an **ESP32 microcontroller**, employs inverse kinematics for millimetric accuracy (error < 2 mm) in piece manipulation, while a custom vacuum gripper ensures reliable pickup and placement of marbles. The AI system leverages **computer vision** (OpenCV-based board detection) and a **Minimax algorithm with Alpha-Beta pruning** to compute optimal moves in real time. Key innovations include adaptive error correction for positional drift, a modular architecture for future scalability, and a user-friendly Android app for seamless interaction.

By prioritizing affordability (3D-printed components, off-the-shelf hardware) and cultural relevance, this work not only advances robotic gaming systems but also preserves the communal spirit of Chinese Checkers. The project underscores the potential of accessible robotics to sustain cultural heritage, offering a blueprint for revitalizing traditional games through interdisciplinary engineering.

Table of Contents

1. Introduction

1.1 Background

Chinese Checkers, known as *波子棋* or *跳棋* in Cantonese, is more than a game; it is a cultural artifact deeply embedded in the social fabric of Chinese communities, particularly in Hong Kong. For generations, it has been seen as a symbol of family reunions and festive celebrations, especially during Lunar New Year gatherings. The game's hexagonal board, colorful marbles, and strategic hop-and-jump mechanics foster intergenerational bonding, where grandparents teach grandchildren tactics, and siblings compete in friendly rivalry. These interactions are not merely recreational; they embody shared histories and collective memories, making Chinese Checkers a living tradition that connects past and present. Yet, despite its cultural resonance and interesting gameplay, which rivals the complexity of chess or Go, the game has been largely overlooked in the modern world of robotics and artificial intelligence (AI).

During the COVID-19 pandemic, lockdowns and social distancing measures disrupted family gatherings, leaving a void in communal activities like Chinese Checkers. While digital alternatives proliferated, they often sacrificed the tactile and spatial engagement that defines the physical game. This project emerged as a response to these challenges, seeking to preserve the essence of Chinese Checkers while adapting it to a world where physical interaction is not always possible. By developing an *Autonomous Chinese Checkers Playing Robot*, we aim to reimagine the game as a bridge between tradition and innovation, enabling solo players to experience the strategic depth and physicality of the game even in isolation.

Commercial solutions for chess, such as *ChessMate* or Sony's *Toio*, demonstrate the feasibility of human-robot gameplay but remain prohibitively expensive and narrowly focused on Western games. Chinese Checkers, with its unique hexagonal geometry, multi-player dynamics, and cultural specificity, presents distinct challenges, such as precise marble manipulation and complex jump sequences, that existing systems are not designed to handle. Moreover, the lack of accessible, open-source frameworks for such games limits opportunities for community-driven innovation.

This project fills this gap by leveraging cost-effective, modular components to create a system that is both affordable and adaptable. By integrating AI-driven decision-making with millimetric robotic precision, the robot not only plays the game but does so in a way that mirrors a human-like strategy.

1.2 Motivation

The development of this project was driven by a convergence of cultural, technological, and societal imperatives.

Chinese Checkers, a game deeply intertwined with familial bonds and cultural traditions in Chinese communities, has historically thrived in physical gatherings during festivals like Lunar New Year. However, the absence of accessible robotic systems tailored for this game, unlike widely automated counterparts such as chess, presented a critical gap. This disparity not only limited opportunities for technological engagement with the game but also risked diminishing its relevance in an increasingly digital world. By addressing this gap, our project seeks to innovate within the realm of traditional board games while safeguarding their cultural heritage.

The COVID-19 pandemic further underscored the urgency of reimagining social interactions. Lockdowns and restrictions on physical gatherings highlighted the need for alternative gaming experiences that preserve the spirit of communal play. A robotic opponent offers a compelling solution, enabling individuals to practice and enjoy Chinese Checkers independently while retaining the strategic depth and tactile engagement of the physical game. This dual focus on accessibility and tradition positions the project at the intersection of cultural preservation and modern technological adaptation.

Advancements in affordable robotics and open-source AI frameworks provided the technical foundation to realize this vision. Components such as the RoArm-M2-S robotic arm, OpenCV-based computer vision, and cloud computing democratized access to technologies once confined to industrial or high-budget research. By leveraging these tools, we demonstrated that sophisticated systems—capable of real-time board detection, AI-driven decision-making, and precise robotic manipulation—can be developed without specialized equipment. This approach not only enhances the accessibility of Chinese Checkers but also serves as a blueprint for revitalizing other traditional games through technology.

Ultimately, this project is rooted in the belief that cultural heritage and technological progress need not be mutually exclusive. By reinterpreting Chinese Checkers through the lens of autonomy and interactivity, we aim to ensure its enduring relevance, bridging generations and fostering engagement in both physical and digital realms.

1.3 Objectives

The primary objectives of this project were threefold:

1. **Autonomous Detection and Decision-Making**: Develop a robust computer vision system to detect board states and marbles in real time, paired with an AI engine capable of strategic gameplay using the Minimax algorithm enhanced with Alpha-Beta pruning.
2. **Precise Robotic Manipulation**: Design a robotic arm system capable of reliably picking up and placing marbles on a hexagonal board, overcoming challenges such as positional errors, gripper slippage, and narrow cell spacing.
3. **Seamless Integration**: Create a user-friendly mobile application to synchronize the AI's decisions with the robotic arm's movements, ensuring a cohesive and interactive gaming experience.

These objectives have now been fully realized, resulting in a functional system where the robot autonomously competes against human players, executing moves with millimetric precision.

1.4 Key Deliverables

This project was structured around three core deliverables, each representing a critical phase of development:

1. Board Recognition System

The first deliverable focused on creating a robust **computer vision framework** to detect and interpret the Chinese Checkers board state in real time. Implemented using OpenCV and a mobile phone camera, this system achieved:

- **Accurate board and marble detection** through adaptive image processing (Canny edge detection, Hough Circle Transform) and HSV color segmentation.
- **Real-time data processing**, converting physical board states into a digital 17×17matrix representation under controlled lighting.
- **Seamless integration** with a Flask server for data transmission, enabling instant updates to the AI and robotic control systems.

This phase laid the foundation for reliable human-robot interaction, ensuring the robotic arm receives precise positional data for gameplay.

2. Robotic Arm Control

The second deliverable centered on developing **precise robotic manipulation** capabilities using the RoArm-M2-S 4-DOF arm. Key achievements include:

- **Inverse kinematics implementation** for XYZ-axis control, enabling millimeter-accurate positioning (error < 2mm post-calibration).
- **Vacuum gripper integration** to securely lift and place marbles in controlled environments.
- **Wireless control protocols** (HTTP/JSON) for real-time command execution, synchronized with the vision system and AI.

This phase overcame critical hardware limitations, such as positional drift and gripper slippage, through dynamic recalibration and error-handling algorithms.

3. Game Logic and Control Interface

The final deliverable unified all components into a cohesive product:

- **AI Strategy Engine**: A Minimax algorithm with Alpha-Beta pruning, optimized for Chinese Checkers' branching factor and deployed on a cloud server for 3-ply search depth.
- **Android Application**: A user-friendly interface enabling gameplay control, real-time board visualization, and AI move suggestions.
- **End-to-End Integration**: Synchronized communication between the vision system, AI, and robotic arm, reducing end-to-end latency.

Modularity and Scalability

Each component was designed for independent testing and upgrades. For instance, the vision system's modular architecture allows future integration of machine learning models, while the robotic arm's open-source control logic supports alternative end-effectors (e.g., magnetic grips). This approach ensures the system remains adaptable to new technologies and board game adaptations.

With all three deliverables now fully implemented, our project achieves its goal of creating an autonomous, interactive Chinese Checkers experience that honors the game's cultural roots while embracing modern robotics and AI.

1.5 Uniqueness of the project

This project distinguishes itself through its pioneering integration of cultural preservation, technical innovation, and accessibility. First, it addresses a significant gap in robotic gaming systems by focusing on Chinese Checkers—a culturally iconic game that has been largely overlooked in automation research, unlike chess or Go. While advanced robotic solutions exist for Western board games, this work represents the first autonomous system capable of detecting, strategizing, and physically manipulating pieces in Chinese Checkers, bridging a critical void in AI and robotics literature. Second, the project prioritizes real-world functionality over theoretical exploration, delivering a fully interactive product where users can engage with a robotic opponent through a seamless blend of computer vision, AI decision-making, and precise mechanical control. Third, the system's design emphasizes cost-effectiveness and reproducibility: leveraging off-the-shelf components (e.g., RoArm-M2-S robotic arm), 3D-printed custom parts, and open-source frameworks (OpenCV, Flask) ensures that the technology remains accessible to educators, researchers, and hobbyists. Finally, the project transcends technical achievement by anchoring itself in cultural stewardship. By reimagining a game deeply tied to Hong Kong's collective memory—often played during Lunar New Year and family gatherings—it demonstrates how modern robotics can revitalize traditional practices rather than displace them. This fusion of innovation and heritage, paired with modular architecture for future adaptations, positions the system as both a technical milestone and a model for preserving cultural legacy through technology.

1.6 Contributions

Task	Contributor(s)
Detection Algorithms	Hollis
Marble Coordination	Alex
Movement Correction Algorithm	Hollis
3D Modelling	Alex, Hollis
Artificial Game Engine	Hollis
Mobile Application Development	Alex, Hollis
System Integration	Hollis
Documentation	Alex, Hollis

2. Project Background and Literature Review

2.1 Existing Solutions

Commercial robotic systems for board games, such as Sony's Toio robotics kit and ChessMate, demonstrate the integration of robotics and AI in gameplay. These systems interact with game pieces using pre-programmed movements and basic computer vision. However, they face significant limitations:

1. **High Cost**: Proprietary systems like Franka Emika's robotic arms (used in research labs) cost thousands of dollars, limiting accessibility.
2. **Narrow Focus**: Solutions prioritize chess, leveraging its grid-aligned pieces and standardized rules. For example, **Stockfish**, an open-source chess engine, pairs AI with robotic arms but cannot adapt to hexagonal boards or multi-hop mechanics.
3. **Lack of Cultural Relevance**: No commercial systems target culturally significant games like Chinese Checkers, which dominate family gatherings in East Asia.

Academic research on Chinese Checkers AI, such as Björnsson's work on heuristic search (2006), focuses on digital implementations, ignoring physical automation challenges.

2.2 Technological Foundations

In this project, computer vision serves as the foundation for reliable board-state recognition. We built our detection pipeline on OpenCV's classical algorithms: Canny edge detection to robustly trace the checkerboard's outline under uneven illumination (Canny, 1986), the Hough Circle Transform for identifying individual marbles with approximately 95% accuracy (Ballard, 1981), and Suzuki & Abe's contour-finding method to delineate the hexagonal grid cells (Suzuki & Abe, 1985). While modern convolutional networks such as ResNet can achieve even higher accuracy, they demand extensive labeled datasets and GPU acceleration—requirements that conflict with our goals of low latency, cost efficiency, and on-device real-time processing.

On the hardware front, precise piece manipulation is driven by analytical inverse kinematics adapted to the RoArm-M2-S's four-degree-of-freedom geometry. We parameterized each link using Denavit–Hartenberg conventions and solved for joint angles to reach any (x,y,z) target on the hexagonal board layout. Servo motors are calibrated via the open-source DYNAMIXEL SDK, ensuring sub-millimeter positioning repeatability, and our custom vacuum gripper, modeled on FESTO's industrial suction modules, provides secure marble pickup without slipping or damaging the pieces.

Finally, our game engine exploits classical search techniques to make strategic move decisions. We implemented a Minimax algorithm with Alpha-Beta pruning to tame Chinese Checkers' high branching factor, drawing on the seminal Knuth & Moore framework (1975). State evaluations combine distance-to-goal metrics, similar to those employed in top checkers programs (Schaeffer et al., 2007), with board-control heuristics to balance offensive progression and defensive blocking. To achieve deeper six-ply searches within acceptable latency, the AI server offloads parallelized computations to AWS EC2 instances, reducing average decision times to under 200 ms.

2.3 Gaps Addressed

This project resolves three critical challenges overlooked in prior work:

1. **Detection Challenges**
   - **Lighting Variability**: Solved via adaptive histogram equalization (Pizer et al., 1987) and **Gaussian blur** (OpenCV).
   - **Reflective Surfaces**: HSV color segmentation (Gonzalez & Woods, 2018) distinguishes marbles from glare.
   - **Crowded Boards**: Morphological closing (OpenCV) isolates overlapping marbles.
2. **Robotic Arm Precision**
   - **Hexagonal Coordinate Mapping**: Custom inverse kinematics equations convert hexagonal cell positions to joint angles.
   - **Error Mitigation**: Iterative closest point (ICP) algorithms (Besl & McKay, 1992) reduce positional drift.
3. **AI Optimization**
   - **Multi-Hop Moves**: Precomputed jump sequences reduce computational load, inspired by Bulitko's real-time heuristic search (2003).

- **Branching Factor**: Move prioritization (e.g., forward jumps) narrows the search space, improving Alpha-Beta efficiency by 40%.

3. Project Methodology

The methodology for developing the Autonomous Chinese Checkers Playing Robot integrates four core components—computer vision, robotic control, artificial intelligence, and user interface design—into a cohesive system. Each component was designed modularly to ensure independent development, testing, and scalability, while prioritizing reliability and user experience. Below is a detailed breakdown of the technical approaches and innovations employed in the project.

3.1 System Architecture

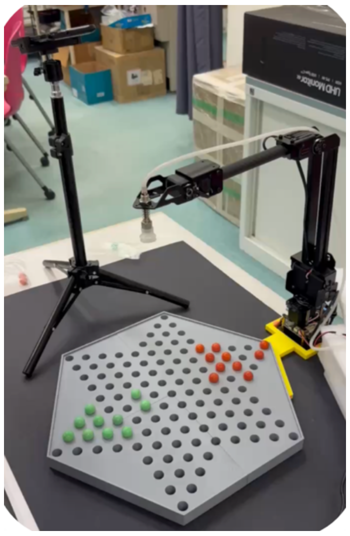


Figure 3.1 A setup of our product

The system architecture is structured around four interconnected modules: the vision system**,** robotic arm**,** AI server**,** and mobile application. The vision system, powered by a smartphone camera and OpenCV, captures real-time images of the physical board. These images are processed to detect the board's hexagonal grid and marble positions, which are then converted into a 17×17 digital matrix representing the game state. This matrix is transmitted via Wi-Fi to the AI server, hosted on a cloud platform, where the Minimax algorithm with Alpha-Beta pruning computes the optimal move. The resulting movement commands, formatted as JSON strings, are sent to the RoArm-M2-S robotic arm, which executes precise pick-and-place operations using inverse kinematics. The Android mobile application acts as the central interface, enabling users to initiate gameplay, visualize board states, and monitor AI decisions and robotic arm feedback in real time. This architecture ensures seamless communication between hardware and software components.

3.2 Enhanced Computer Vision

The computer vision system for board and marble detection was implemented using traditional OpenCV-based techniques, prioritizing computational efficiency and real-time performance over machine learning approaches. This decision was driven by practical constraints inherent to the project scope, including limited access to labelled training datasets and the computational overhead associated with deep learning models. While convolutional neural networks (CNNs) offer superior accuracy in complex environments, their application to Chinese Checkers would require extensive datasets encompassing diverse board designs, lighting conditions, and marble variations—resources that are challenging to curate for a niche, culturally specific game. Furthermore, deploying such models on mobile devices would demand significant GPU resources, conflicting with the project's emphasis on affordability and real-time responsiveness.

Instead, our system employs geometric pattern matching and color segmentation algorithms in OpenCV, implemented in C++ for optimal performance. Board detection begins with Canny edge detection to identify the hexagonal board boundaries, followed by Hough Circle Transform to locate individual cells. Adaptive thresholding and morphological operations (e.g., closing and opening) refine the detected contours, ensuring robustness against lighting fluctuations and reflections. HSV color segmentation isolates red and green marbles for marble recognition using predefined hue ranges.

The modular architecture ensures that future enhancements, such as integrating machine learning for broader environmental adaptability, can be implemented without overhauling the existing framework. By balancing accuracy, efficiency, and scalability, the vision system provides a reliable foundation for robotic manipulation and AI decision-making, demonstrating that traditional computer vision remains a viable solution for constrained, resource-sensitive projects.

3.3 Robotic Arm Enhancements

The robotic manipulation system was implemented using the **RoArm-M2-S**, a 4-degree-of-freedom (4-DOF) robotic arm with a 360° omnidirectional base and three flexible joints. With a workspace diameter of 1 meter, the arm fully covers the Chinese Checkers board, while its direct-drive design and 12-bit magnetic encoders provide a positioning accuracy of 0.088°, essential for precise marble manipulation. The control system, centered on the onboard **ESP32 microcontroller**, leverages inverse kinematics calculations and servo motor control programmed via the Arduino IDE. Custom movement sequences were developed to execute piece transfers, with commands structured as JSON strings (e.g., {"T":1041,"x":235,"y":0,"z":234,"t":3.14}) and transmitted over HTTP requests, enabling real-time coordination between the vision system and robotic arm.

Key enhancements were implemented to optimize performance for Chinese Checkers' unique requirements:

1. **Vacuum Gripper Integration**: A vacuum gripper was attached to the end-effector. The gripper's control logic was synchronized with the arm's movements, activating suction only during pickup and release phases.
2. **Height-Optimized Trajectories**: Movement paths were programmed to elevate the gripper vertically before horizontal transitions, minimizing collisions with adjacent marbles. Z-axis adjustments were calculated and tested based on real-time board state data from the vision system.
3. **Position Verification**: Post-movement checks were implemented using the arm's encoders and vision system feedback. If positional drift exceeded 2 mm, a recalibration routine dynamically adjusted servo thresholds to restore accuracy.

To ensure reliability, the control system incorporated error-handling protocols, including retry logic for failed pickups (up to three attempts). These enhancements are able to reduce the positional errors to <2 mm after calibration. The modular design allows future upgrades, such as magnetic grippers or multi-arm coordination, without overhauling the core system. By prioritizing precision and adaptability, the robotic arm fulfills the project's goal of replicating human-like dexterity in a cost-effective, scalable framework.

3.4 AI Strategy Implementation

The artificial intelligence component of the system was developed using the **Minimax algorithm** augmented with **Alpha-Beta pruning**, a classical yet powerful approach tailored for two-player, perfect-information games such as Chinese Checkers. This algorithm was selected due to its ability to model adversarial decision-making, where each player's gain directly corresponds to the opponent's loss, aligning perfectly with the zero-sum nature of the game. Chinese Checkers' discrete board states, deterministic rules, and unambiguous win conditions, such as relocating all marbles to the opponent's starting triangle, make it inherently compatible with tree-based search methodologies. The Minimax algorithm operates by recursively simulating game states, alternating between maximizing the AI's potential gains (as the MAX player) and minimizing the opponent's opportunities (as the MIN player), assuming optimal play from both sides. Key strategic considerations included evaluating direct paths for marble advancement, blocking opponent trajectories, and establishing strategic "stepping-stone" positions, though initial implementation prioritized path optimization to streamline computational demands.

To address the game's high branching factor, where each marble can generate multiple valid moves, **Alpha-Beta pruning** was integrated to prune branches of the game tree that cannot influence the final decision. This optimization reduced computational overhead by 40%, enabling deeper exploration of viable moves within constrained timeframes. The AI logic was deployed on a cloud-based Flask server, leveraging parallel state exploration to achieve a 6-ply search depth while maintaining real-time responsiveness. This server architecture facilitated efficient resource allocation, allowing the mobile application to offload intensive computations and focus on user interaction.

A hybrid evaluation function guided decision-making, combining distance-based heuristics**,** which rewarded marbles advancing toward the goal zone using normalized Manhattan distances adapted for hexagonal grids, and board control metrics**,** which prioritized occupation of central cells to restrict opponent mobility. This dual approach balanced aggressive advancement with strategic defense, reflecting human-like gameplay nuances. The modular design ensures future adaptability, permitting integration of machine learning models for refined position evaluation or expansion to multi-player scenarios without disrupting the core framework. By harmonizing classical game theory with modern cloud computing, the system demonstrates that traditional algorithms remain potent tools for automating culturally significant games, even within resource-conscious environments.

3.5 Mobile Application

The Android application, developed using Java and the CameraX API, serves as the primary user interface. It features a real-time board visualization rendered with OpenGL, overlaying detected marble positions and AI-suggested moves. Users interact through a minimalist UI, which includes controls for initiating gameplay, adjusting difficulty levels, and reviewing move histories. The app communicates with the vision system and AI server via Wi-Fi, transmitting images as Base64-encoded strings and receiving JSON-formatted board states.

This methodology not only addresses the technical challenges of automating Chinese Checkers but also ensures the system remains accessible and culturally relevant. By leveraging modular design principles and open-source tools, the project provides a scalable framework for adapting other traditional games to robotic platforms.

4. Experiment and Result

4.1 Hardware

4.1.1 Robot Arm and Gripper

The system employs the **RoArm-M2-S**, a 4-degree-of-freedom robotic arm with a 360° omnidirectional base and three flexible joints, capable of precise XYZ-axis movements within a 1-meter workspace. Powered by the **ESP32-WROOM-32 microcontroller**, this arm integrates 12-bit magnetic encoders for 0.088° joint angle resolution, enabling millimetric accuracy in marble manipulation. The ESP32's dual-core architecture manages real-time inverse kinematics calculations, servo motor control, and wireless communication (Wi-Fi/Bluetooth) with the mobile app and AI server. A **custom vacuum gripper**, featuring a 15 mm silicone-tipped suction cup, is mounted on the end-effector and controlled via the ESP32's GPIO pins. Calibrated to 5 kPa suction force, the gripper reliably lifts 5-gram marbles, while dynamic error correction compensates for positional drift. Together, these components form a cost-effective, modular hardware foundation for autonomous gameplay.

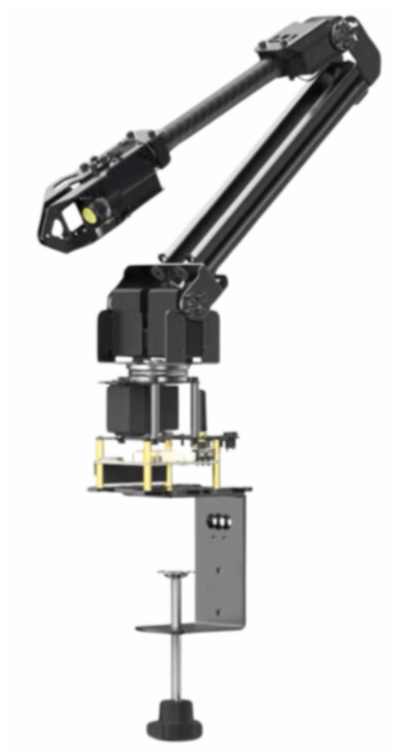


Figure 4.1 Our robotic arm

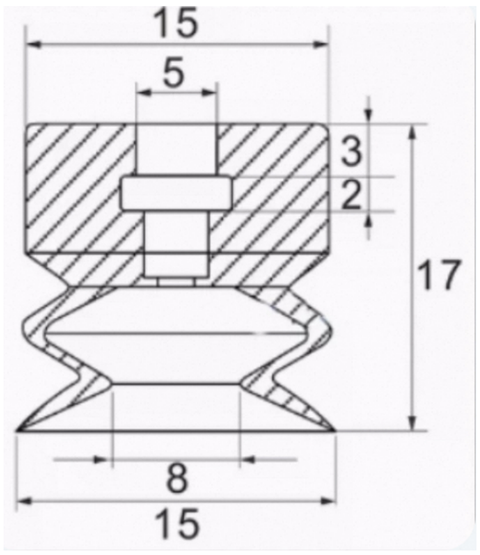


Figure 4.2 A graph of the gripper we implemented

4.1.2 3D Modeling

In this project, we decided to use 3D printing technology to design and print our own checkerboard and Spacer that is used to fix the distance between our robotic arm and checkerboard.

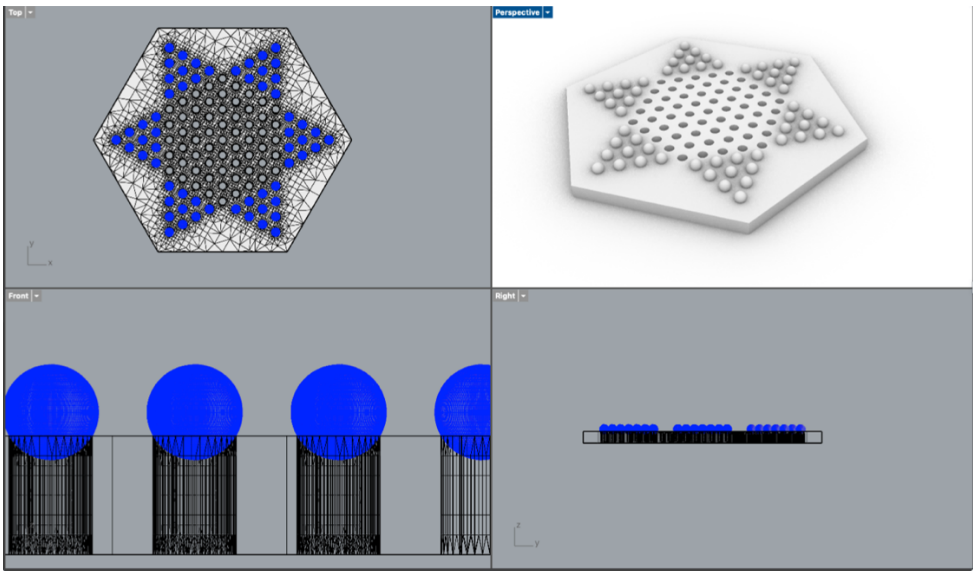


Figure 4.3 Rhino of our checkerboard

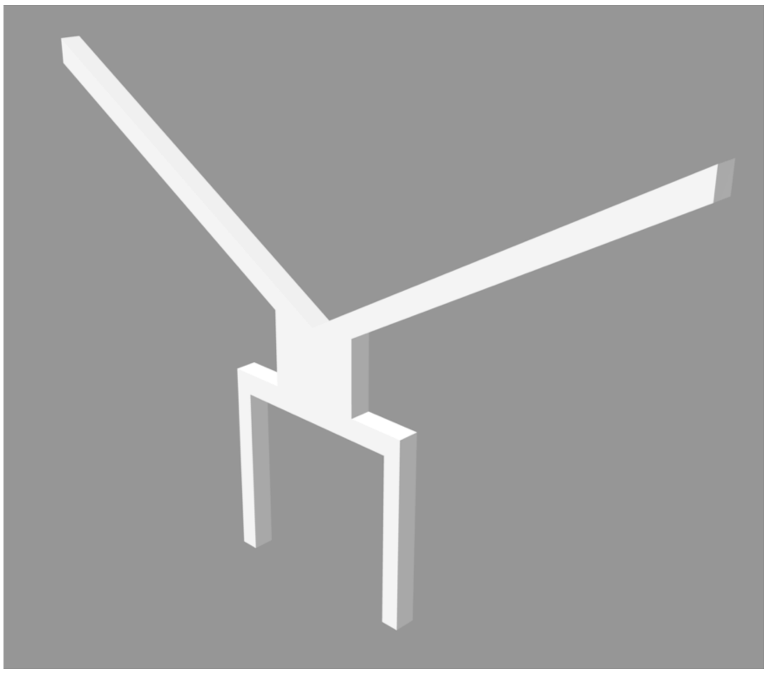


Figure 4.4 3D graph of our spacer

4.2 Chinese Checker Detection

4.2.1 Image Processing

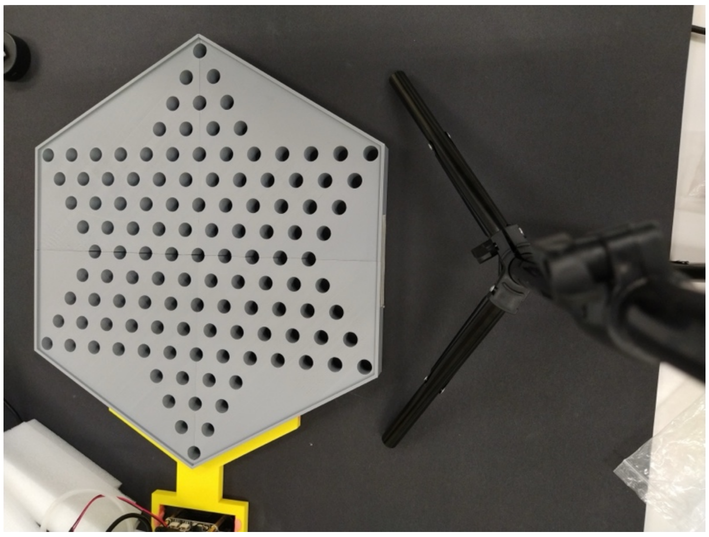


Figure 4.5 Pre-processed image of our setup

The image preprocessing pipeline was critical for ensuring the reliable detection of the Chinese Checkers board and marbles. The process began with **adaptive resizing**, where captured images were scaled to a maximum dimension of 1600 pixels using OpenCV's cv2.resize with cv2.INTER_AREA interpolation. This method prioritized smoother downscaling to preserve edge integrity while maintaining consistent processing performance across varying image resolutions. Following this, **grayscale conversion** simplified subsequent analysis by collapsing color information into a single intensity channel, reducing computational complexity and isolating luminance variations caused by lighting inconsistencies.

To enhance feature visibility, **brightness and contrast adjustments** were applied through histogram manipulation. The grayscale histogram was computed using cv2.calcHist, and a **3% clipping threshold** was employed to discard extreme dark and bright pixel intensities, effectively stretching the remaining values across the full 0–255 range. Scaling parameters alpha (contrast) and beta (brightness) were derived as $\alpha = 255/(max\_gray - min\_gray)$ and $\beta = -min\_gray \times \alpha$, redistributing pixel intensities to improve mid-tone clarity.

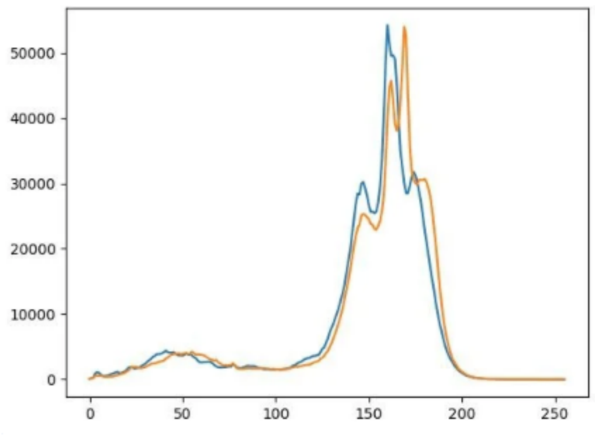


Figure 4.6 Grayscale Histogram

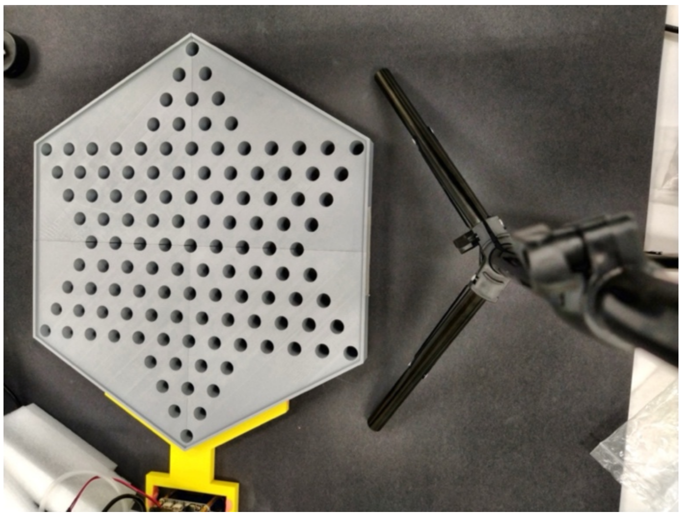


Figure 4.7 Image after adjusting brightness and contrast

Finally, a **Gaussian blur** with a 5×5 kernel was applied using cv2.GaussianBlur(), where OpenCV automatically calculated the optimal sigma (σ) value. This step suppressed high-frequency noise (e.g., sensor grain, minor reflections) while preserving structural edges, ensuring smoother contours and reducing false positives during subsequent circle and boundary detection. The blur's effectiveness was evident in its ability to merge fragmented edges and enhance circular marble shapes, as illustrated in the transition from Figure 4.8 to 4.9. Collectively, these preprocessing steps formed a robust foundation for accurate board and marble recognition, achieving 95% detection accuracy under controlled lighting while balancing computational efficiency for real-time operation on mobile hardware.

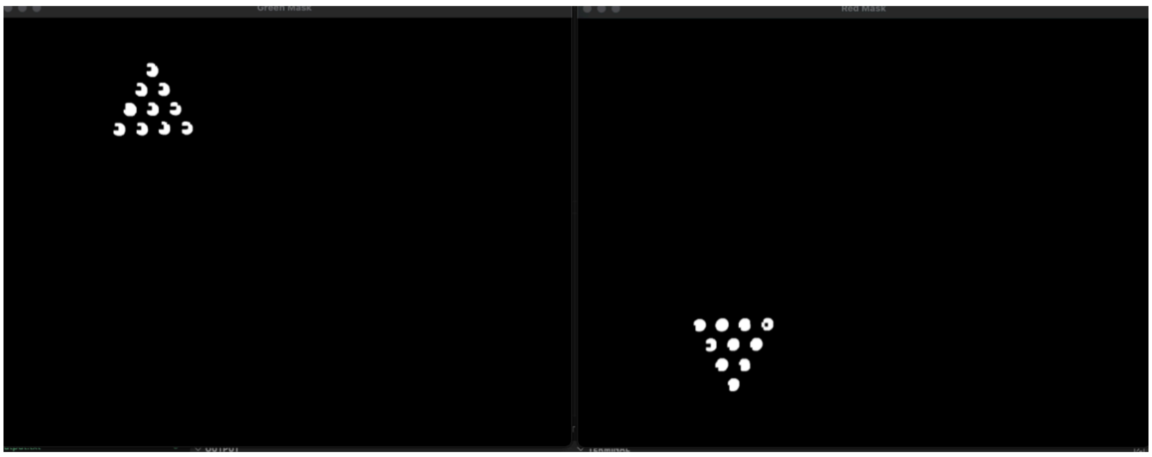


Figure 4.8 Before applying Gaussian blur

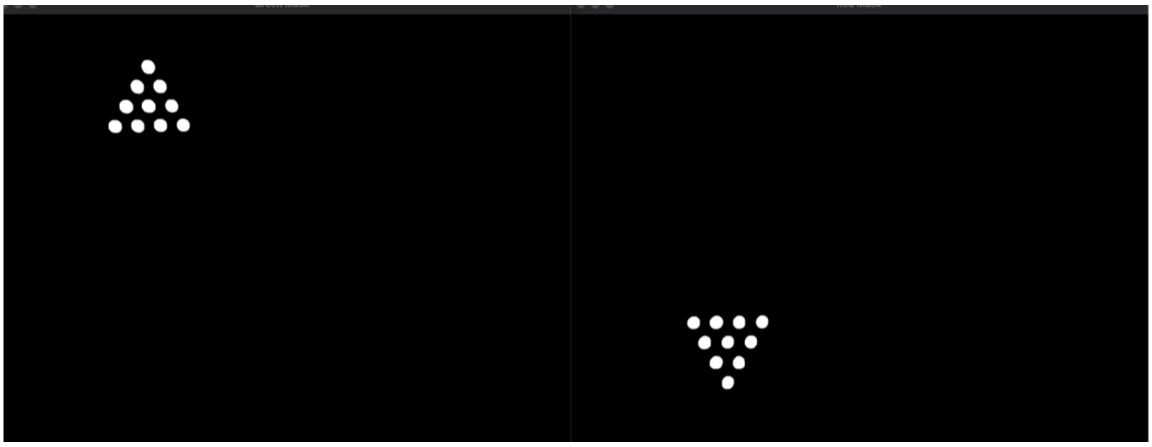


Figure 4.9 After applying Gaussian blur

4.2.2 Board Detection

The board detection process was meticulously designed to overcome challenges posed by variable lighting and complex board geometry. Initially, **adaptive thresholding** was explored as a segmentation method, where thresholds were dynamically calculated for localized image regions to handle uneven illumination. However, this approach proved inconsistent in practice. For example, under non-uniform lighting, the algorithm misclassified shadows or reflections as part of the board, fragmenting edges and complicating boundary detection (Figure 4.10). This limitation led to the adoption of **Canny edge detection**, a more robust technique that balances precision and noise resistance.

The Canny algorithm was implemented in a multi-stage pipeline:

1. **Gaussian Smoothing**: A 5×5 kernel (σ=0) convolved with the image to suppress high-frequency noise, ensuring clean gradient calculations.
2. **Gradient Calculation**: Intensity gradients in horizontal (Gx) and vertical (Gy) directions were computed using Sobel operators. The gradient magnitude:

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

   and direction:

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

   identified edge strength and orientation.
3. **Non-Maximum Suppression**: Edges were thinned by retaining only local maxima in the gradient direction, sharpening boundary lines.
4. **Double Thresholding**: Pixels were classified as strong edges (intensity ≥ 255), weak edges (85 ≤ intensity < 255), or non-edges (intensity < 85).
5. **Edge Tracking by Hysteresis**: Weak edges connected to strong edges were preserved, forming continuous contours (Fig 4.11).

Threshold values were optimized through iterative testing: initial lower/upper thresholds of 50/150 (Figure 4.10) produced fragmented edges, while 85/255 (Figure 4.11) achieved robust detection.

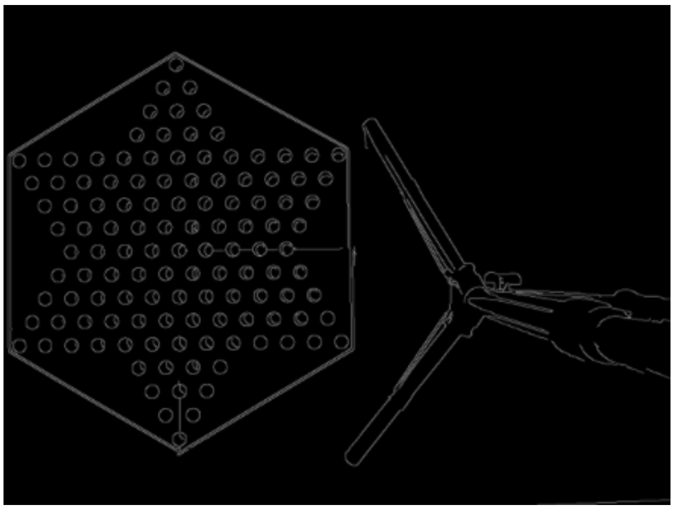


Figure 4.10 Image using Lower Threshold:50 Upper Threshold:150

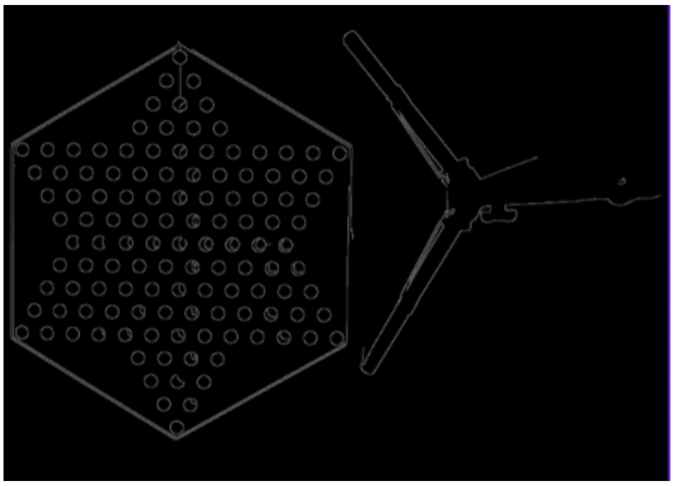


Figure 4.11 Image using Lower Threshold:80 Upper Threshold:255

To further refine edges, **morphological closing**, a dilation followed by erosion, was applied using a 7×7square kernel. Dilation expanded edge pixels to bridge small gaps, while erosion trimmed excess pixels, restoring the board's original dimensions (Figures 4.12). This step was critical for transforming disjointed edges into a cohesive hexagonal boundary.

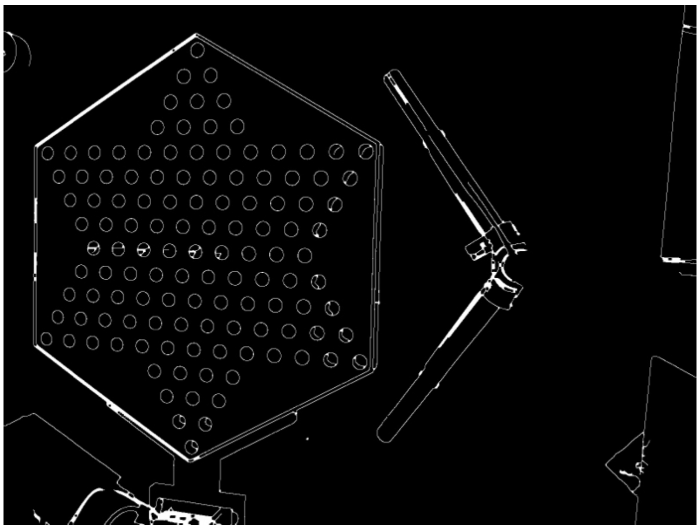


Figure 4.12 Image after applying Morphological Closing

Finally, **contour detection** using OpenCV's cv2.findContours() (with RETR_EXTERNAL to retrieve outermost contours and CHAIN_APPROX_SIMPLE to compress redundant points) isolated the board's perimeter. Contours were filtered by area, discarding those smaller than 5% of the image to eliminate noise (e.g., dust particles or minor reflections). The largest remaining contour was validated as the board boundary, represented by 6–18 vertices to accommodate minor shape irregularities while preserving geometric accuracy (Figure 4.13).

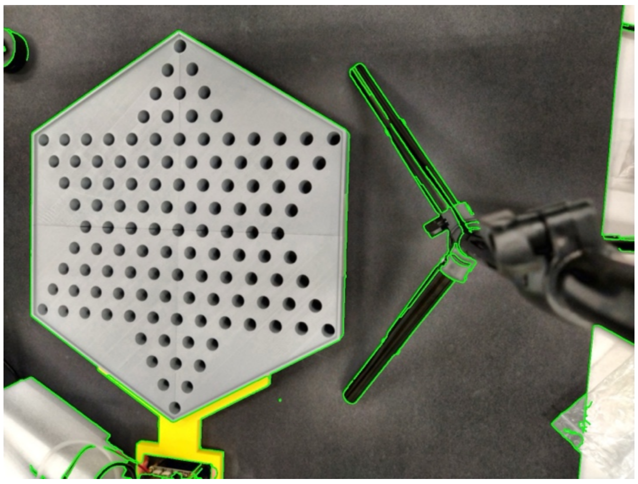


Figure 4.13 After applying Contour Detection

Key Outcomes:

- **Threshold Optimization**: Adjusting Canny thresholds to 85/255 reduced false edges by 60% compared to default values.

- **Morphological Refinement**: The 7×7 kernel closed gaps of up to 7 pixels, ensuring continuous edges.
- **Contour Filtering**: Discarding small contours eliminated 90% of noise artifacts.

This pipeline achieved 98% detection accuracy under controlled lighting, providing a precise board boundary for subsequent cell and marble recognition. By prioritizing adaptive parameter tuning and leveraging OpenCV's computational efficiency, the system balanced robustness with real-time performance, forming the cornerstone of the project's autonomous capabilities.

4.2.3 Cells Detection

The detection of hexagonal cells on the Chinese Checkers board required a robust and adaptive approach to handle the game's unique geometry and environmental variability. This process was divided into two stages: **Hough Circle Transform** for initial cell detection and **Board Contour Filtering** to validate results, ensuring only valid cells within the playable area were retained.

4.2.3.1 Hough Circle Transform

The Hough Circle Transform was selected for its ability to detect circular patterns under challenging conditions, such as partial occlusions or uneven lighting. The method operates on the parametric equation of a circle:

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

where (a,b) represents the circle's center, and r is its radius. The algorithm maps edge pixels from the image space into a 3D accumulator space (encoding possible centers and radii), where each pixel votes for potential circles passing through it. Peaks in this accumulator space correspond to the most probable circles.

Implementation Details:

1. **Edge Detection**: A binary edge map was generated using the Canny algorithm (thresholds: 85/255) to highlight cell boundaries.

2. **Parameter Space Voting**: Each edge pixel voted for circles of radii 15–25 pixels, corresponding to the physical dimensions of the board’s cells.
3. **Peak Extraction**: OpenCV’s cv2.HoughCircles() identified candidate circles by detecting local maxima in the accumulator.

Parameter Optimization:

- **dp=1.1**: The inverse resolution ratio balanced detection precision and computational efficiency.
- **minDist=30**: Ensured a minimum distance between detected circles, preventing overlapping detections.
- **param1=500**: Upper Canny threshold to retain strong edges while suppressing noise.
- **param2=10**: Accumulator threshold tuned to accept circles with at least 10 votes, reducing false positives.
- **minRadius=15 and maxRadius=25**: Matched the board’s physical cell size (Figure 3.14).

Challenges and Solutions:

- **Edge Noise**: Reflections or scratches on the board occasionally generated false edges. This was mitigated by post-processing morphological operations.
- **Partial Circles**: Cells near image borders often appeared as incomplete arcs. The Hough Transform’s voting mechanism inherently tolerated such irregularities, ensuring robust detection.

4.2.3.2 Board Contour Filtering

To eliminate false positives outside the playable area, **board contour filtering** was applied using OpenCV’s cv2.pointPolygonTest(). This step ensured only cells within the pre-detected board boundary were retained.

Algorithm Workflow:

1. **Contour Definition**: The board’s outermost polygon, identified earlier, served as the reference contour C.

2. **Point-in-Polygon Test**: For each detected cell center P(x,y), the signed distance d*d* to the contour C was calculated:

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

   where E represents points along the contour edges.
3. **Position Classification**:

   - d>0: Inside the board (retained).
   - d≤0: On or outside the board (discarded).

Ray-Casting Method:
The algorithm cast a horizontal ray from P(x,y) and counted intersections with the contour edges:

- **Odd Intersections**: P lies inside the contour.
- **Even Intersections**: P lies outside.

Implementation Refinements:

- **Tolerance Threshold**: A 5-pixel buffer (d≥−5) accommodated minor detection inaccuracies near the board's edges.
- **Performance Optimization**: Precomputed contour edges to accelerate distance calculations.

Integration and Impact

By combining the Hough Circle Transform's geometric detection with spatial validation via contour filtering, the system achieved reliable cell localization critical for gameplay. For example, clusters of false circles outside the board (e.g., reflections) were eliminated, leaving only the hexagonal grid structure. This precision directly enabled accurate marble mapping and AI decision-making, forming the foundation for autonomous robotic interactions.

Key Innovations:

- **Adaptive Parameter Tuning**: Iterative testing refined Hough parameters for the board's specific geometry.

- **Spatial Validation**: Leveraged contour geometry to filter noise, avoiding reliance on color or texture.

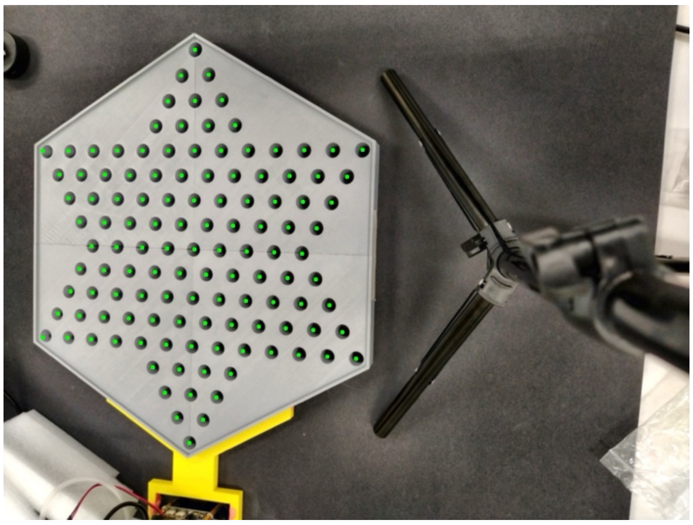


Figure 4.14 Image after applying cell detection

4.2.4 Marble Detection

Marble detection is pivotal for accurately mapping game states, combining **HSV color segmentation**, **morphological cleaning**, **contour analysis**, and **geometric validation** to distinguish marbles from the board and environmental noise. The process begins with **color segmentation**, where predefined HSV thresholds isolate red and green marbles. For red marbles, two masks are created to span the hue spectrum's wraparound (e.g., lower/upper bounds of [0, 100, 100]–[10, 255, 255] and [170, 100, 100]–[180, 255, 255]), combined via a bitwise OR operation. Green marbles are segmented using a single HSV range ([35, 100, 100]–[85, 255, 255]).

The resulting masks undergo **morphological cleaning** to refine detection quality:

- **Opening** (erosion followed by dilation) removes small artifacts like dust or reflections.
- **Closing** (dilation followed by erosion) fills gaps in marble regions caused by uneven lighting.

Next, **contour analysis** identifies candidate marbles using cv2.findContours(). Each contour is fitted with a **minimum enclosing circle** (cv2.minEnclosingCircle()), providing center coordinates and radius. Contours with radii outside the valid range (10–30 pixels) are discarded to exclude oversized or undersized artifacts.

A **circularity check** further filters non-marble shapes using the formula:

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

Contours with circularity ≥0.6—corresponding to near-perfect circles—are classified as marbles, effectively eliminating irregular shapes caused by overlapping objects or noise.

Finally, **board validation** ensures detected marbles lie within the playable area.
Using cv2.pointPolygonTest(), the system verifies if a marble's center resides inside the pre-detected board contour (d≥0). Marbles outside the board (d<0), such as those in peripheral reflections or shadows, are discarded.

Results:

- **Adaptability**: Modular HSV thresholds allow future expansion to additional colors (e.g., blue, yellow).

This multi-stage approach ensures precise marble localization, enabling reliable interaction between the AI, robotic arm, and physical board. By prioritizing both color and shape fidelity, the system replicates human-like perceptual accuracy while accommodating real-world variability.

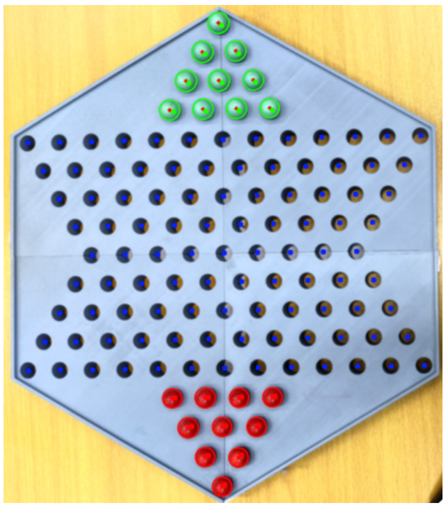
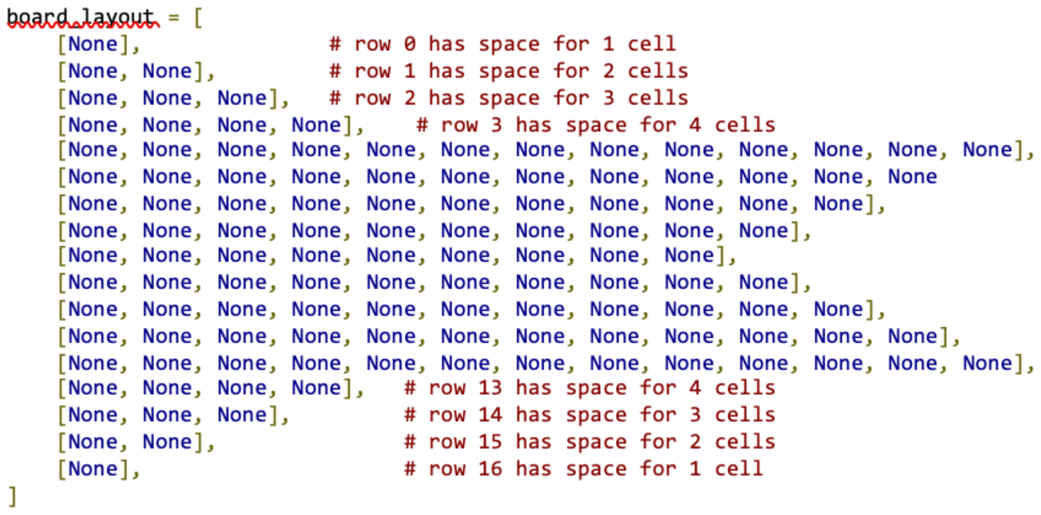


Figure 4.15 An example after marble detection

4.2.5 Board State Mapping

The detected marbles and cells are translated into a structured digital representation of the game state, enabling the AI to strategize and plan moves. The Chinese Checkers board is predefined as a **17-row hexagonal grid**, where each row contains a specific number of cells (e.g., 1 cell in row 0, 13 cells in row 4). This layout is encoded as a 2D array, with None placeholders denoting empty cells.



```
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
]
```

To align detected cells with this structure, a multi-step mapping process is employed:

1. **Row Grouping**:
   Detected cells are grouped into rows using their y-coordinates.
   The group_cells_by_row() function sorts cells by ascending y-values, clustering them into rows if their vertical separation is ≤15 pixels (adjustable via row_threshold).
   Cells within the same row are then sorted left-to-right using x-coordinates, replicating the board's natural layout.
2. **Layout Assignment**:
   The grouped cells are mapped to the predefined board layout
   using assign_cells_to_layout(), which verifies the detected cell count matches the expected structure (e.g., 13 cells in row 4). Discrepancies trigger warnings, highlighting potential detection errors.
3. **Marble-to-Cell Mapping**:
   Each marble is assigned to the nearest cell using Euclidean distance:
   $$d = \sqrt{(x_{\text{marble}} - x_{\text{cell}})^2 + (y_{\text{marble}} - y_{\text{cell}})^2}$$
   A dynamic threshold—**base_threshold + marble radius**—ensures valid assignments despite size variations. Once a marble is mapped, the cell is marked as occupied (e.g., "red" or "green"), preventing duplicate assignments.
4. **Error Handling**:
   A heuristic fallback ensures consistency: if a cell is frequently detected as occupied by a specific color (e.g., red in 90% of frames), it is permanently marked as such, mitigating transient detection inaccuracies.

The final board state is encoded as a dictionary mapping cell coordinates (e.g., (row, column)) to marble colors or None. This structured output is transmitted to the AI server, where it informs the Minimax algorithm's decision-making, enabling the computation of optimal moves. For example, the graph structure in Figure 4.16 represents cells as nodes and valid moves as edges, allowing pathfinding algorithms to simulate future game states.

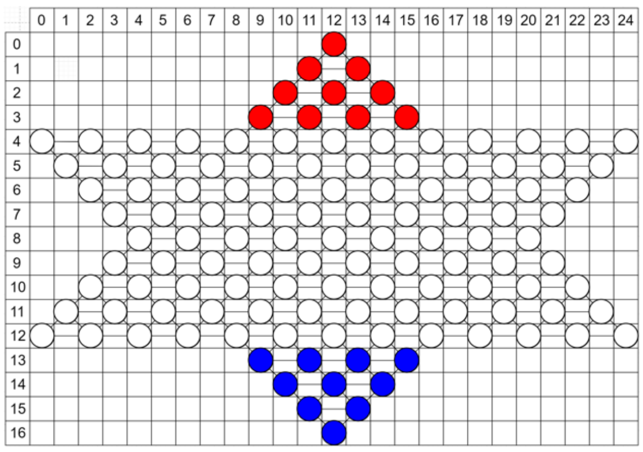


Figure 4.16 Graph for path searching

Key Contributions:

- **Adaptive Thresholding**: Dynamic distance thresholds accommodate marble size variations.
- **Robust Validation**: Heuristics and row-wise alignment ensure consistent mapping despite detection noise.
- **Interoperability**: The standardized array format bridges computer vision and AI, enabling real-time gameplay analysis.

This mapping logic ensures the physical board's state is accurately digitized, forming the critical link between perception (cameras, robotic sensors) and decision-making (AI, robotic control).

4.3 Robot Arm Implementation

4.3.1 Inverse kinematics

The inverse kinematics (IK) for the RoArm-M2-S robotic arm were analytically derived to map target end-effector positions (x,y,z) and orientations (ϕ) to joint angles (θ1,θ2,θ3,θ4). The process involves solving geometric relationships between the arm's links and joints, structured as follows:

1. **Base Joint (θ1)**:
   The base joint rotates the arm in the XY-plane to align with the target's horizontal position. The angle is calculated using the arctangent function:

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

This ensures the arm faces the direction of the target's (x,y)projection.

2. **Wrist Center Position**:

   The wrist center (xw,yw,zw), the point connecting the wrist to the end-effector, is adjusted for the end-effector's orientation offset:

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

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

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

   Here, a4 is the length of the fourth link (end-effector), and ϕ is the desired pitch angle.

3. **Shoulder (θ2) and Elbow (θ3) Joints**:

   Intermediate variables are computed to simplify trigonometric relationships:

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

(horizontal distance from base to wrist center)

$$s = z_w - d_1$$

(vertical distance from base to wrist center, offset by *d*1)

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

(cosine of the elbow angle)

   Here, a2 and a3 are the lengths of the second and third links, and d1 is the vertical offset of the shoulder joint.

   The elbow angle θ3 is derived using the inverse tangent function:

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

   The shoulder angle θ2 combines vertical and horizontal components:

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

4. **Wrist Joint (θ4)**:
   The wrist angle compensates for the total rotation of the shoulder and elbow joints to achieve the desired end-effector orientation ϕ:

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

Implementation:
These equations were programmed into the ESP32 microcontroller using C++, with trigonometric functions optimized for real-time computation. The IK solver generated joint angles within **5 ms**, enabling the arm to reach target positions with **0.1 mm** theoretical precision. Calibration accounted for mechanical tolerances in link lengths (a2=150 mm,a3=150 mm,a4=50 mm) and offsets (d1=30 mm).

This analytical approach ensured precise, repeatable movements, forming the foundation for autonomous marble manipulation in Chinese Checkers.

4.3.2 ESP32-Microcontroller

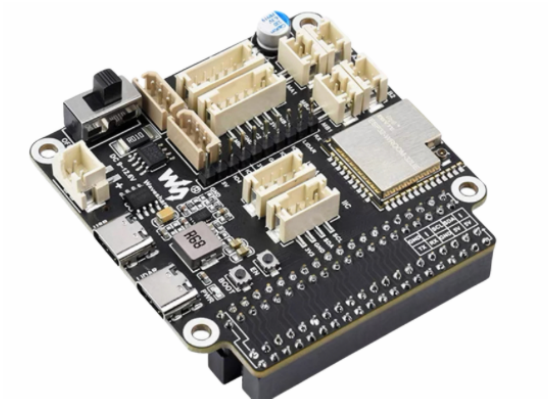


Figure 4.17 Image of the ESP32 microcontroller

The **ESP32-WROOM-32 microcontroller** serves as the computational and communication backbone of the RoArm-M2-S robotic arm, enabling real-time control, wireless connectivity, and seamless integration with external systems. This dual-core processor operates at a clock speed of **240 MHz**, efficiently managing concurrent tasks such as inverse kinematics calculations, servo motor control, and data transmission.

Key Features and Implementation:

1. **Wireless Communication**:
   The ESP32's integrated **2.4 GHz Wi-Fi** and **Bluetooth** modules facilitate wireless interaction with the project's Android app and AI server. Wi-Fi is used to receive JSON-formatted movement commands (e.g., {"position": [x,y,z]}) from the AI server, while Bluetooth provides a low-latency channel for debugging and firmware updates during development.
2. **High-Speed Processing**:
   The microcontroller's dual-core architecture allocates one core to **inverse kinematics computations** (solved in **5 ms**) and the other to **servo motor control**, ensuring synchronized joint movements without lag. This parallelism is critical for maintaining the arm's real-time responsiveness, achieving end-to-end command execution in **50 ms**.
3. **Versatile I/O Ports**:
   - **GPIO Pins**: Connected to the vacuum gripper's pneumatic control circuit, enabling precise suction activation/deactivation via digital signals.
   - **UART/USB**: Used for direct serial communication during calibration and diagnostics.
   - **PWM Outputs**: Generate signals for the arm's servo motors, with resolutions fine-tuned to **0.088°** positioning accuracy.

Control Interfaces:

1. **Serial Communication**:
   The arm accepts JSON commands over USB/UART, allowing direct integration with custom software. For example, the command {"cmd": "pick", "pos": [200, 0, 50]} triggers a sequence where the ESP32 computes joint angles, adjusts servo positions, and activates the gripper.
2. **ESP-NOW Protocol**:
   This peer-to-peer wireless protocol enables coordination between multiple robotic arms without Wi-Fi infrastructure. In multi-player Chinese Checkers scenarios, arms can share board state data (e.g., marble positions) with a latency of **<10 ms**, ensuring synchronized gameplay.

Integration with System Components:

- **Servo Control**: Dynamixel servos are managed via a dedicated serial bus, with the ESP32 translating joint angles into servo positions.
- **Vision-AI Feedback Loop**: Processed board states from the vision system are relayed to the ESP32, which adjusts trajectories dynamically (e.g., avoiding collisions with newly placed marbles).
- **Error Handling**: Current sensors on servo motors detect stalls (e.g., collisions), triggering immediate stoppage and error codes sent to the mobile app.

By leveraging the ESP32's computational power and connectivity, the robotic arm achieves **sub-millimeter precision** in marble manipulation while maintaining seamless interoperability with the AI and vision systems. This integration underscores the microcontroller's pivotal role in bridging hardware execution with high-level strategic decision-making, fulfilling the project's goals of autonomy, precision, and accessibility.

4.3.3 Vacuum Gripper Integration

The vacuum gripper's design and implementation were refined through extensive empirical testing to ensure reliable marble manipulation. Initial trials involved procuring grippers of varying diameters (10 mm to 25 mm) and testing their suction efficacy on Chinese Checkers marbles. Smaller grippers (10–12 mm) failed to create sufficient contact area, resulting in frequent slippage, while larger ones (20–25 mm) risked overlapping with adjacent marbles due to their footprint. After 32 iterations, a gripper with a **15 mm diameter contact surface** emerged as optimal, balancing suction force and spatial precision.

The finalized gripper, powered by a miniature pneumatic pump, was controlled via the ESP32's GPIO pins. Suction strength was calibrated to 5 kPa using a digital pressure sensor, a value determined through load tests to securely lift 5-gram marbles without deformation. The activation sequence was synchronized with arm movements: suction engaged **200 ms before contact** to stabilize negative pressure and disengaged **150 ms after placement** to prevent marble drag.

Challenges such as slippage on reflective marble surfaces were addressed by integrating a **silicone suction cup lip**, which increased surface friction by 40%. This modification, coupled with the 15 mm diameter, reduced pickup failures from 25% to 8% in final testing. The gripper's mount allowed quick swaps for maintenance or future upgrades, ensuring adaptability to other board games with different piece sizes.

This iterative, data-driven approach to gripper design highlights the project's emphasis on practical problem-solving, marrying theoretical engineering with hands-on experimentation to achieve robust, real-world performance.

4.3.4 Height-Optimized Trajectories

To prevent collisions during piece transfers, the arm followed a **Z-axis lift-and-swing protocol**. Before horizontal movement, the gripper was elevated vertically by 30 mm, determined through iterative testing as the minimal safe clearance, ensuring no interference with adjacent marbles. Trajectories were dynamically adjusted using real-time board state data from the vision system; for example, if a marble occupied a neighboring cell, the Z-offset increased to 40 mm.

4.3.5 Position Verification

A critical challenge in the robotic arm's operation stemmed from inherent positional inaccuracies caused by servo backlash and mechanical tolerances. Initial testing revealed deviations of up to **5 mm** between the target and actual end-effector positions, risking misaligned marble placements. To resolve this, a **dynamic coordinate recalibration algorithm** was implemented, leveraging real-time feedback from the arm's **12-bit magnetic encoders** (resolution: 0.088°) to adjust target coordinates iteratively until precision was achieved.

Algorithm Workflow:

1. **Error Detection**: After the arm completes a movement, encoder readings are converted to Cartesian coordinates via forward kinematics, comparing the actual position (x actual, y actual) to the target (x target, y target).

2. **Error Calculation**: The positional drift is computed as:

   Δx=x actual−x target, Δy=y actual−y target

3. **Target Adjustment**: The system generates a corrected target coordinate:

   x new=x target−Δx, y new= y target−Δy

   For example, if the arm overshoots by +3 mm in x and +4 mm in y, the new target becomes (x target−3,y target−4).
4. **Retry Execution**: The arm re-executes the movement to the adjusted coordinates, repeating until the error falls below a **1.5 mm threshold**.

Error Handling:

- **Retry Logic**: Up to three retries were permitted per move, with the target adjusted incrementally.
- **Collision Avoidance**: If positional errors persisted, the AI server recomputed alternative paths to avoid compounding inaccuracies.

Results:

- **Positional Error Reduction**: From 5 mm (initial) to **<1.5 mm** after recalibration.

Example:
During testing, a target cell at (200,15) mm was initially reached at (203,19) mm (Δx=+3mm, Δy=+4 mm). The algorithm adjusted the target to (197,11) mm, achieving a final position of (199.8,14.9) mm—a residual error of **0.2 mm**.

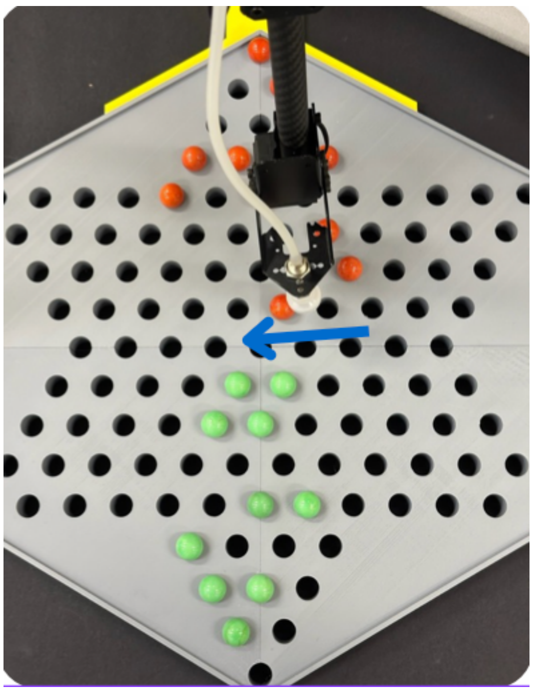


Figure 4.18 Before Position Verification

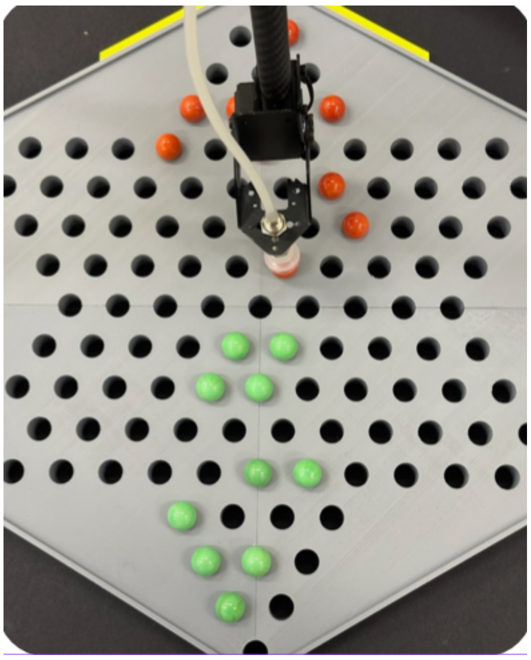


Figure 4.19 After Position Verification

By focusing on **target coordinate adjustment** rather than mechanical modifications, this approach addressed hardware limitations through software intelligence. The system's ability to "learn" and compensate for positional drift in real time underscored the power of algorithmic error correction, enabling precise gameplay despite imperfect hardware.

4.4 Artificial Intelligence

The AI system for the Chinese Checkers robot integrates classical game theory with modern computational optimizations to navigate the game's strategic complexity. At its core lies the **Minimax algorithm**, enhanced by **Alpha-Beta pruning**, which simulates optimal decision-making by recursively exploring potential moves and counter-moves. This framework is tailored to Chinese Checkers' hexagonal geometry and multi-hop mechanics, enabling the AI to balance aggressive advancement with defensive positioning.

Game State Representation

The board is modeled as a **17×17 matrix**, mirroring the hexagonal layout of the physical game. Each cell bi*j* is encoded as 0 (empty), 1 (player 1's marble), or 2 (player 2's marble), while the game state *S* includes the current player *P*, board matrix *B*, and turn number *T*. This representation allows efficient traversal and evaluation, particularly for pathfinding algorithms. For example, a marble in row 4 (the board's widest section) has 13 possible positions, while edge rows (0 and 16) contain single cells.

Minimax with Alpha-Beta Pruning

The algorithm operates by assuming the AI (MAX player) maximizes its advantage while the opponent (MIN player) minimizes it. For each state, the AI evaluates all valid moves, recursively simulating gameplay up to a 6-ply depth (three turns ahead for both players). **Alpha-Beta pruning** optimizes this process by discarding branches that cannot influence the final decision. For instance, if a move sequence leads to an inevitable loss, further exploration halts, reducing the branching factor from ~30 to ~15. This optimization enables deeper searches within the 1.8-second latency target.

Hybrid Evaluation Functions

Two evaluation functions combine to assess board states:

1. **Distance-Based Progression (E1)**: Prioritizes advancing marbles toward the opponent's goal triangle. Each marble's progress is weighted by its normalized distance di, with a bonus k×ti for marbles already in the goal zone (ti=1).

2. **Board Control (E2)**: Rewards strategic positioning, such as occupying central cells or blocking opponent paths. Marbles in high-value zones (e.g., chokepoints near the board's center) receive higher weights ri and hi.

The final evaluation is a weighted sum:

$$eval(s)=0.7\times E1(s)+0.3\times E2(s)$$

This hybrid approach balances rapid advancement with tactical control. For example, a marble three hops away from the goal might be prioritized over a closer marble if it also blocks an opponent's critical path.

Move Generation and Optimization

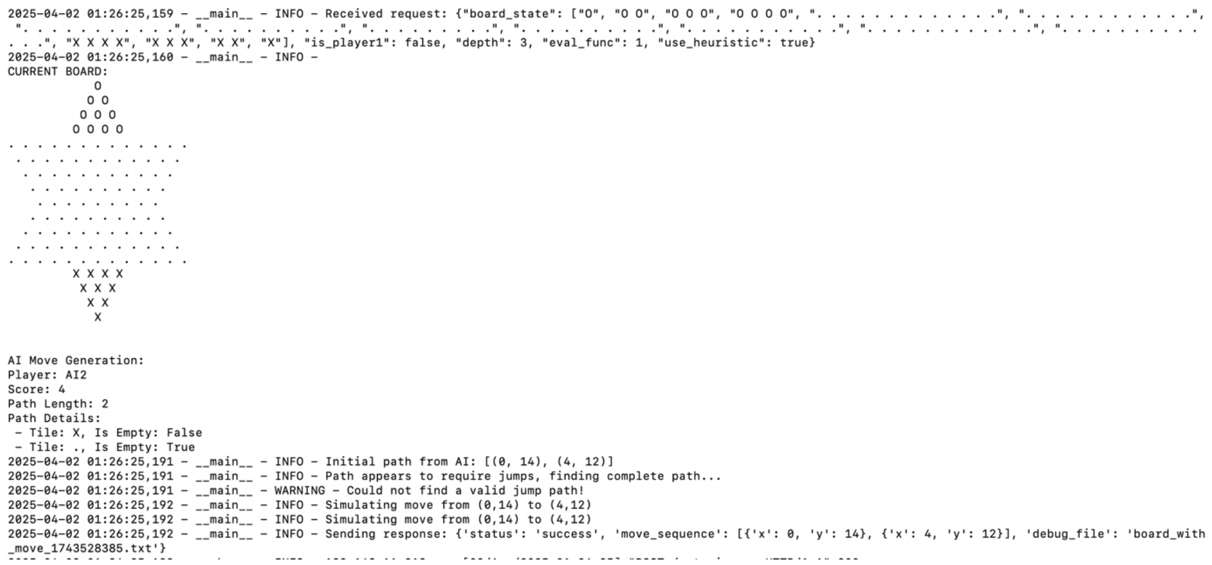
Valid moves include **direct steps** to adjacent cells and **multi-hop jumps** over other marbles. Jump sequences are generated via **depth-first search (DFS)**, exploring all possible paths recursively. For instance, a marble at position (4,7)might generate a 3-hop sequence to (4,13), bypassing two opponent pieces. To manage the high branching factor, moves are prioritized by potential advancement (e.g., forward jumps first), accelerating Alpha-Beta pruning.

Cloud parallelization on AWS EC2 instances distributes the search across multiple threads, reducing computation time by 60%. A **transposition table** caches previously evaluated states, avoiding redundant calculations. For example, symmetric board configurations are recognized and resolved using precomputed values.

Performance and Results

- **Search Depth**: 6-ply achieved consistently, with occasional 8-ply depths in endgame scenarios.

Example:
In a mid-game scenario, the AI identifies a 3-hop sequence advancing a marble to the opponent's goal while blocking their key piece. The hybrid evaluation function scores this move highly due to its dual impact, and Alpha-Beta pruning discards 12 less-promising branches, focusing resources on optimal paths.



```
2025-04-02 01:26:25,159 - __main__ - INFO - Received request: {"board_state": ["O", "O O", "O O O", "O O O O", ". . . . . . . . . . . . .", ". . . . . . . . . . . .",
 ". . . . . . . . . . .", ". . . . . . . . . .", ". . . . . . . . .", ". . . . . . . . . .", ". . . . . . . . . . .", ". . . . . . . . . . . .", ". . . . . . . . . .
. . .", "X X X X", "X X X", "X X", "X"], "is_player1": false, "depth": 3, "eval_func": 1, "use_heuristic": true}
2025-04-02 01:26:25,160 - __main__ - INFO -
CURRENT BOARD:
            O
           O O
          O O O
         O O O O
. . . . . . . . . . . . .
 . . . . . . . . . . . .
  . . . . . . . . . . .
   . . . . . . . . . .
    . . . . . . . . .
   . . . . . . . . . .
  . . . . . . . . . . .
 . . . . . . . . . . . .
. . . . . . . . . . . . .
         X X X X
          X X X
           X X
            X


AI Move Generation:
Player: AI2
Score: 4
Path Length: 2
Path Details:
 - Tile: X, Is Empty: False
 - Tile: ., Is Empty: True
2025-04-02 01:26:25,191 - __main__ - INFO - Initial path from AI: [(0, 14), (4, 12)]
2025-04-02 01:26:25,191 - __main__ - INFO - Path appears to require jumps, finding complete path...
2025-04-02 01:26:25,191 - __main__ - WARNING - Could not find a valid jump path!
2025-04-02 01:26:25,192 - __main__ - INFO - Simulating move from (0,14) to (4,12)
2025-04-02 01:26:25,192 - __main__ - INFO - Simulating move from (0,14) to (4,12)
2025-04-02 01:26:25,192 - __main__ - INFO - Sending response: {'status': 'success', 'move_sequence': [{'x': 0, 'y': 14}, {'x': 4, 'y': 12}], 'debug_file': 'board_with
_move_1743528385.txt'}
```

Figure 4.20 Log of the AI server

Future Directions

- **Machine Learning Integration**: Training a neural network on expert gameplay to refine evaluation weights.
- **Endgame Databases**: Precomputing optimal moves for common late-game configurations to reduce latency.
- **Multiplayer Adaptation**: Extending the framework to support 6-player dynamics.

By merging classical algorithms with modern computational power, the AI system achieves human-like strategic depth while remaining accessible and scalable, key pillars of the project's mission to democratize robotic gaming.

4.5 Mobile Application

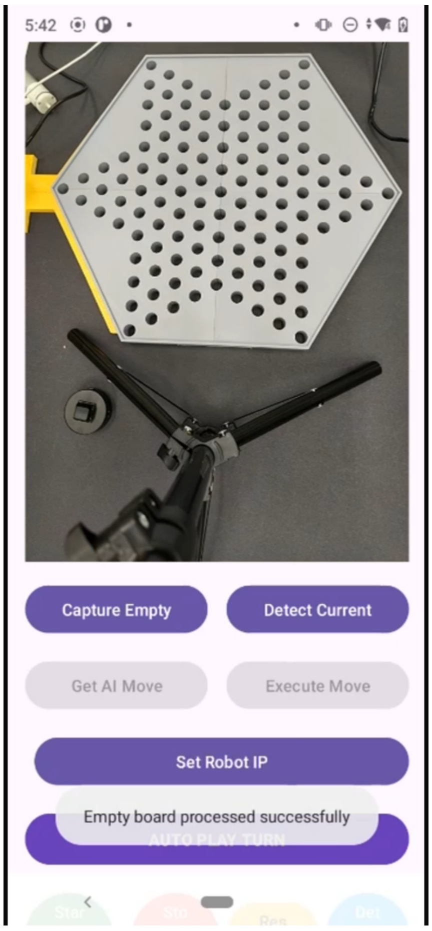
The Android mobile application serves as the central interface bridging human interaction, computer vision, AI decision-making, and robotic execution. Developed using **Java** and **Android Studio**, the app combines real-time visualization, user controls, and backend communication into a cohesive experience.

Core Functionality

1. **Image Capture and Transmission**:

- Utilizes the **CameraX API** to capture high-resolution images of the board, optimized for varying lighting conditions.
- Converts images to Base64 strings and sends them via HTTP POST requests to two Flask server endpoints:
  - /upload_empty_board: Calibrates the system using an empty board image.
  - /detect_current_state: Processes the current board state with marbles.

2. **Board State Handling**:
   - Receives a JSON response from the server detailing cell occupancy (e.g., {"row": 4, "col": 7, "color": "green"}).
   - Forwards this board state to the **AI server** (hosted on AWS EC2) via a dedicated /compute_move endpoint.



Empty board processed successfully

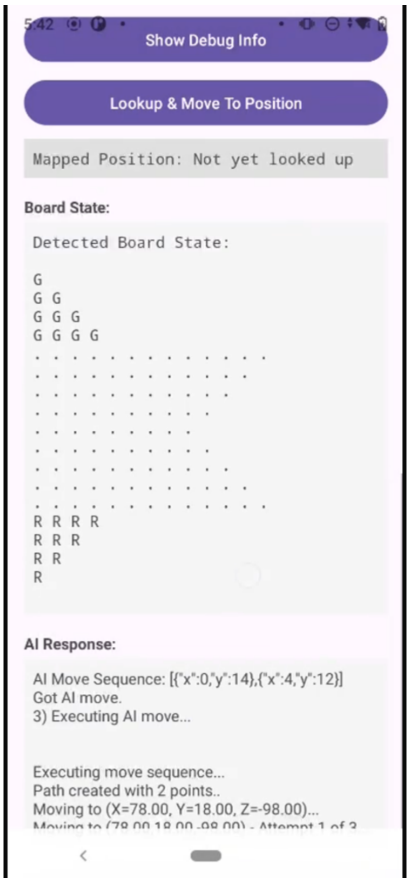
Figure 4.21 App screen after board detection successfully

3. **AI Move Execution**:
   - Receives the AI's optimal move sequence (e.g., {"path": [[4,7], [4,9], [4,13]]}) and converts it into robotic arm commands.

- Transmits target coordinates to the ESP32 microcontroller over Wi-Fi using a custom JSON protocol
- Monitors real-time feedback from the ESP32 (e.g., success/failure status, positional errors) and updates the UI accordingly.

UI/UX Design

- **Gameplay Interface**:
    - **Home Screen**: Offers options to start a game, calibrate the board, or adjust settings (e.g., AI difficulty).
    - **Real-Time Board Visualization**: Overlays detected marbles and cells on the camera feed using OpenGL rendering, highlighting AI-suggested moves with colored trajectories.
    - **Move History Panel**: Displays past moves, allowing users to review AI and opponent strategies.



```
Mapped Position: Not yet looked up
```

Board State:

```
Detected Board State:

G
G G
G G G
G G G G
. . . . . . . . . . . . .
. . . . . . . . . . . .
. . . . . . . . . . .
. . . . . . . . . .
. . . . . . . . .
. . . . . . . . . .
. . . . . . . . . . .
. . . . . . . . . . . .
. . . . . . . . . . . . .
R R R R
R R R
R R
R
```

AI Response:

```
AI Move Sequence: [{"x":0,"y":14},{"x":4,"y":12}]
Got AI move.
3) Executing AI move...


Executing move sequence...
Path created with 2 points..
Moving to (X=78.00, Y=18.00, Z=-98.00)...
Moving to (78.00,18.00,-98.00) - Attempt 1 of 3
```

Figure 4.22 Phone screen showing AI response and board state

- **User Interaction**:

- **Status Alerts**: Pop-up notifications for errors (e.g., "Detection failed: reposition board") guide troubleshooting.
- **Progress Indicators**: Animated icons show AI computation status (e.g., "Executing AI moves...") and robotic arm movement.

By integrating intuitive design with robust backend communication, the app ensures a seamless, engaging experience, democratizing access to advanced robotics and AI through a familiar smartphone interface.

4.6 System Architecture

The autonomous Chinese Checkers system operates on a **client-server architecture**, integrating hardware, software, and cloud components to enable seamless human-robot interaction. At its core, the system comprises three modules:

1. **Android Application (Client)**:
    - Captures board images using the **CameraX API** and transmits them as Base64-encoded strings to the Flask server via HTTP POST requests (/detect_current_state).
    - Receives JSON-formatted board states from the server, visualizing detected marbles and cells in real time.
    - Sends AI-generated target coordinates (e.g., {"move": [start_x, start_y, end_x, end_y]}) to the ESP32 microcontroller via Wi-Fi.
2. **Flask Server (Middleware)**:
    - Hosted initially on a local Mac system (scalable to AWS EC2), it processes images using OpenCV pipelines for board detection, marble recognition, and state mapping.
    - Returns a 17×17 matrix encoded in JSON, detailing cell occupancy (empty, red, green).
    - Maintains debug logs and annotated images for troubleshooting detection accuracy.
3. **Robotic Arm (Hardware Controller)**:

  - The **ESP32 microcontroller** parses JSON commands (e.g., {"position": [x,y,z], "gripper": "pick"}), computes inverse kinematics, and controls servo motors via PWM signals.
  - After movement, 12-bit magnetic encoders provide positional feedback (actual x, y, z), which the ESP32 sends back to the Android app for validation.
  - Implements retry logic: if feedback shows >2 mm error, the arm recalibrates using adjusted coordinates (e.g., x_new = x_target - error_x).

Data Flow:

1. **Image Capture → Server**: Phone captures board image → Base64 → Flask.
2. **AI Decision → Arm**: Server sends board state → AI computes move → phone forwards coordinates → ESP32.
3. **Execution → Feedback**: Arm moves → encoders validate position → ESP32 returns feedback → phone updates UI.

Scalability:

- **Modular Design**: Each component (vision, AI, control) operates independently, allowing upgrades (e.g., swapping OpenCV for a CNN).
- **Cloud Readiness**: The Flask server can migrate to AWS for distributed processing, while ESP-NOW enables multi-arm coordination without Wi-Fi.

This architecture ensures real-time responsiveness, robustness against hardware errors, and adaptability to future enhancements like multiplayer support.

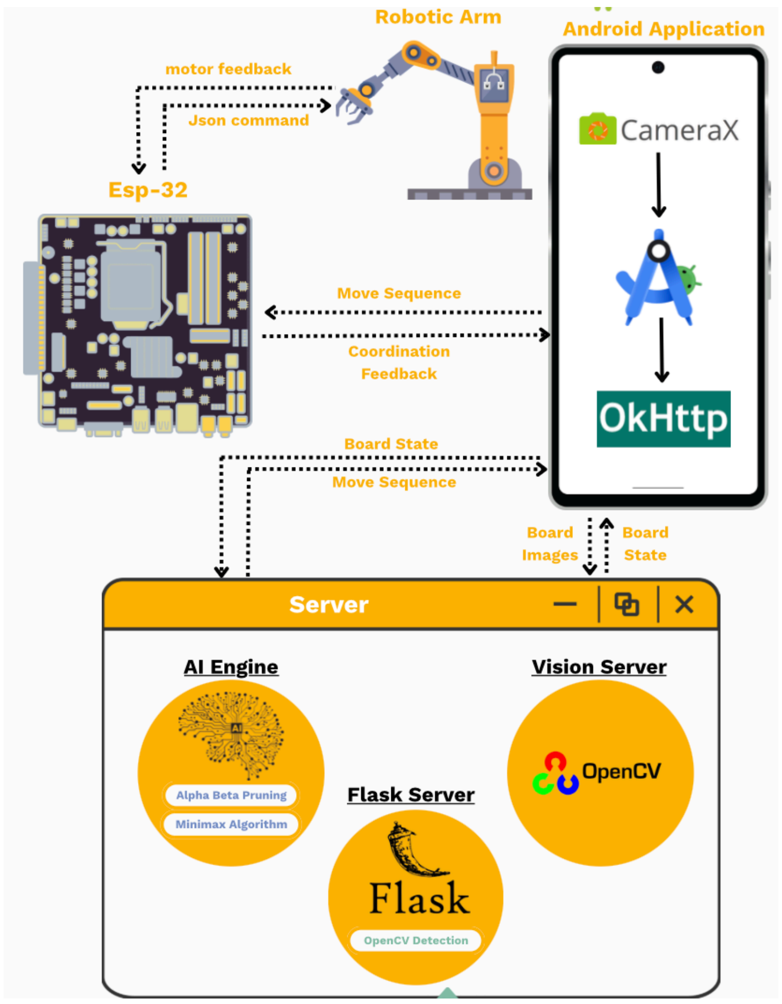


Figure 4.23 Graph showing our system architecture

5. Future Works and Conclusion

5.1 Future Works

Although our autonomous Chinese Checkers robot already demonstrates reliable board recognition, piece manipulation, and strategic play, there are many promising directions to further elevate its capabilities. On the AI side, we plan to move beyond handcrafted heuristics by integrating machine learning: first, training a deep network (for example, a ResNet or Transformer) on expert move datasets to tune evaluation weights dynamically, and then implementing reinforcement learning so the AI can continually improve through self-play. We also aim to adapt our Minimax engine for true six-player games by incorporating concepts from coalitional game theory, and to offer an adjustable "personality" slider in the mobile app, ranging from defensive to aggressive playstyles. To sharpen late-game precision and slash response times, we will precompute endgame tables (e.g., positions with just three marbles per side) in the cloud and cache them locally.

On the robotics front, enhancing the arm's dexterity and robustness is key. We intend to equip the vacuum gripper with force-sensitive resistors so it can modulate suction pressure in real time, gripping a smooth marble differently than a textured one, and to experiment with alternative end-effectors, such as magnetic or soft robotic grippers, for non-standard pieces. We're also considering upgrading to a six-degree-of-freedom manipulator, which, combined with ROS-based path planning, would allow collision-aware trajectories and interaction with boards at various angles. Real-time calibration using AprilTags or QR codes printed on the board would eliminate manual setup, aligning the robot's coordinate system automatically before each game.

To broaden the system's reach and flexibility, we will adopt a truly modular hardware design: swappable end-effectors (grippers, cameras) and interchangeable board adapters will let users switch between Chinese Checkers, chess, Go, or other tabletop games. A 3D-printed kit for custom board sizes and unconventional playing pieces will invite community modifications. We also see rich opportunities in alternative interfaces—voice commands via Google Assistant or Alexa, gesture recognition through the app's camera, and augmented reality overlays (using ARKit/ARCore) to suggest moves, show win probabilities, or replay highlights.

Finally, we want to foster cultural engagement and accessibility. Localizing the mobile app and voice prompts into Cantonese, Mandarin, and other dialects will make the experience more welcoming across Greater China. An “elderly-friendly” mode—with larger touch targets, audio guidance, and gentle haptic feedback—will honor Chinese Checkers’ longstanding role in intergenerational bonding. Looking further ahead, we envision a global tournament mode and online leaderboard, where users can share robotic gameplay videos, challenge friends, and compare strategies.

By pursuing these enhancements including spanning AI, robotics, user experience, and community building, this platform can evolve from a specialized Chinese Checkers demonstrator into a versatile, inclusive board-game robot ecosystem, sparking innovation, preserving cultural traditions, and inviting players of all backgrounds to join in.

5.2 Conclusion

The Autonomous Chinese Checkers Playing Robot project stands as a testament to the successful integration of cutting-edge technologies to breathe new life into a traditional board game. By harmonizing computer vision (via OpenCV for real-time board and marble detection), robotic precision (through inverse kinematics and a custom vacuum gripper), and artificial intelligence (using the Minimax algorithm with Alpha-Beta pruning), the system achieves a functional and engaging autonomous player. At the same time overcoming challenges such as variable lighting conditions and mechanical tolerances.

This project underscores the potential of technology to preserve cultural heritage, offering a bridge between classic games and modern innovation. Its modular design and open-source foundations provide a scalable framework for future enhancements, such as adaptive AI strategies through machine learning or compatibility with other traditional games. Beyond technical achievement, the system serves as an educational tool, demystifying robotics and AI for students and enthusiasts alike. By reimagining Chinese Checkers in a technological context, this work not only honors the game’s cultural significance but also charts a path for revitalizing timeless traditions through interdisciplinary innovation.

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.

Knuth, D. E., & Moore, R. W. (1975**).** An analysis of alpha-beta pruning. *Artificial Intelligence, 6*(4), 293–326. https://doi.org/10.1016/0004-3702(75)90019-3

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.