Chinese Checkerbotic

Autonomous Chinese Checkers Playing Robot Arm

FYP24057

Leung Ho Ning 3035801453
Shiu Chun Nam Alex 3035800849

01

Objectives and Background

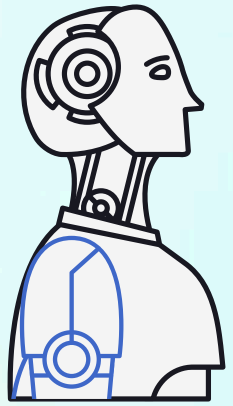
Background

- Rise of AI and Robotics in Chess Games

AlphaGo

OpenCV

- Increased Interest in Robotics



- Cultural Significance

Motivation

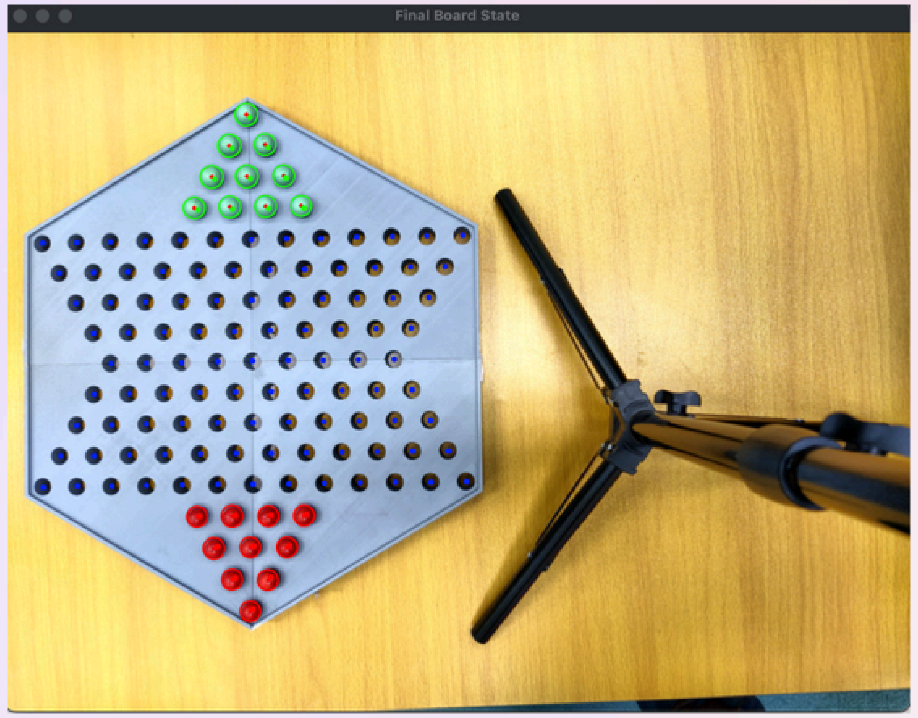
- Nostalgia Meets Innovation
- Bridging the Gap For Chinese Checker compared to other Chess Games

Objectives

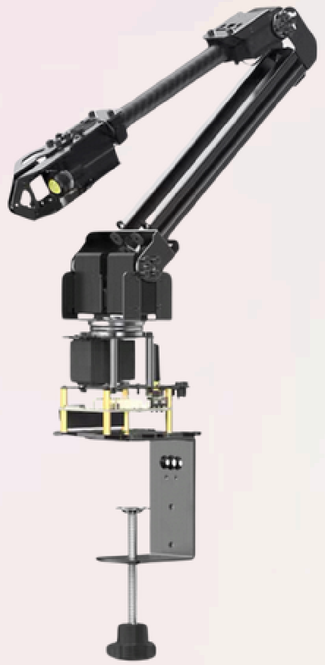
- Innovative Integration
- Enhancing the Gaming Experience

Key Deliverables

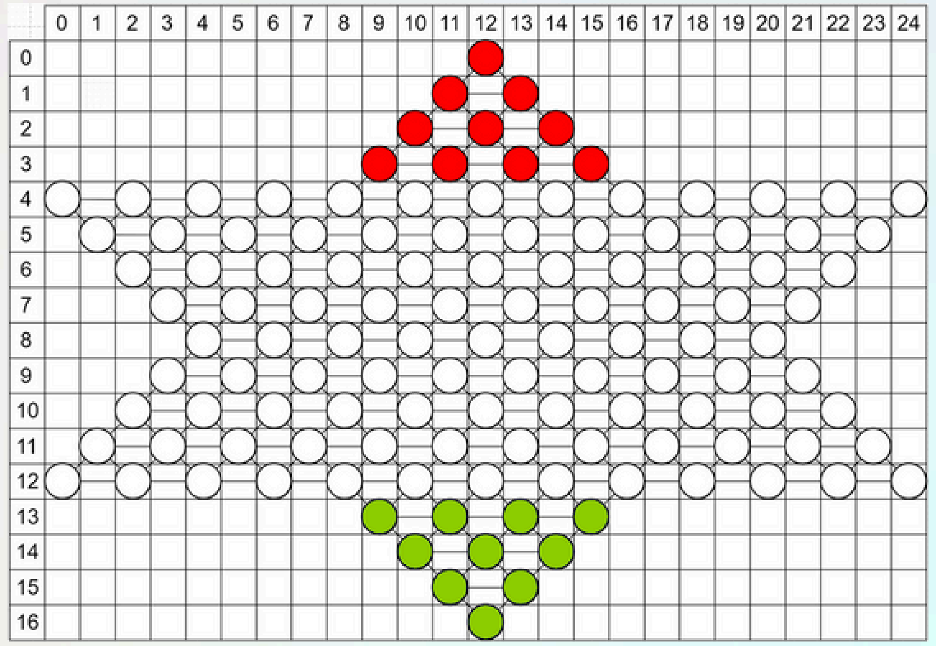
Checker Board Recognition



Remote Control of Robotic Arm



Artificial Chinese Checker



Chinese Checkerbotic

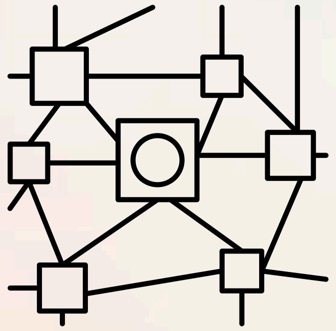
Application Central Hub

Uniqueness

- No existing solutions focus on Chinese Checker detections

- First Automated Player:
  No product exists that can autonomously play Chinese Checkers

- We focus on building a practical system to provide a real-world interactive product.



Preserving its cultural value.

02

Chinese Checker Recognition

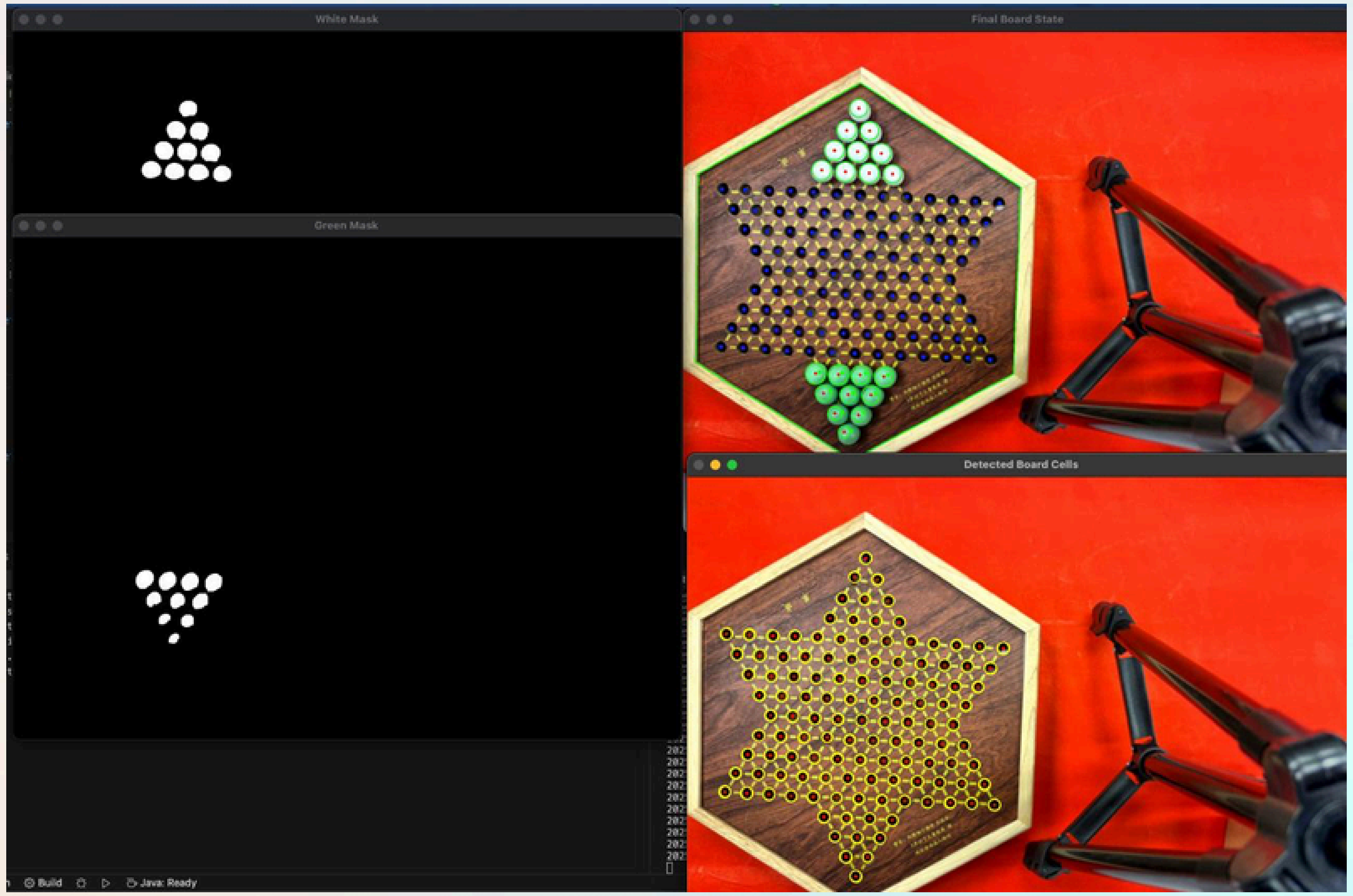
Challenges - Detection

- **Totally algorithmic, Self-design**

Not using ML, don't have data, time, resources...
Not a project on ML but a product
Completely modularized, could enhance in future.

- **Low Tolarance**

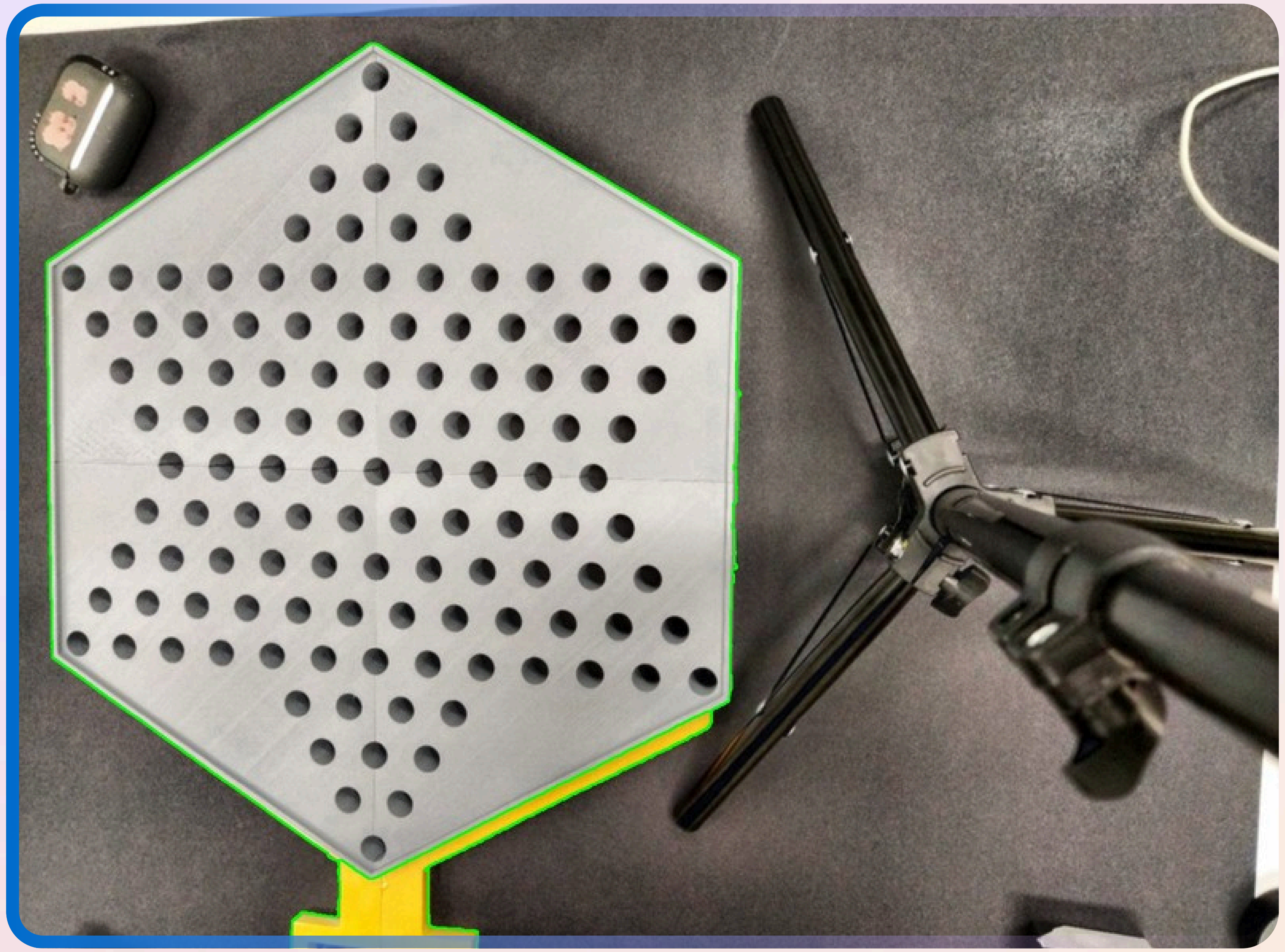
Distances between marbles are too narrow



2.1

Board Detection

Computer Vision - *Board Detection*



Overall Process

Canny Edge Detection
- To highlight the boundaries of shapes like the board's edges.

Morphological Closing
- Fills small gaps in the edges to form a continuous boundary to ensure contours are well-defined for detection.

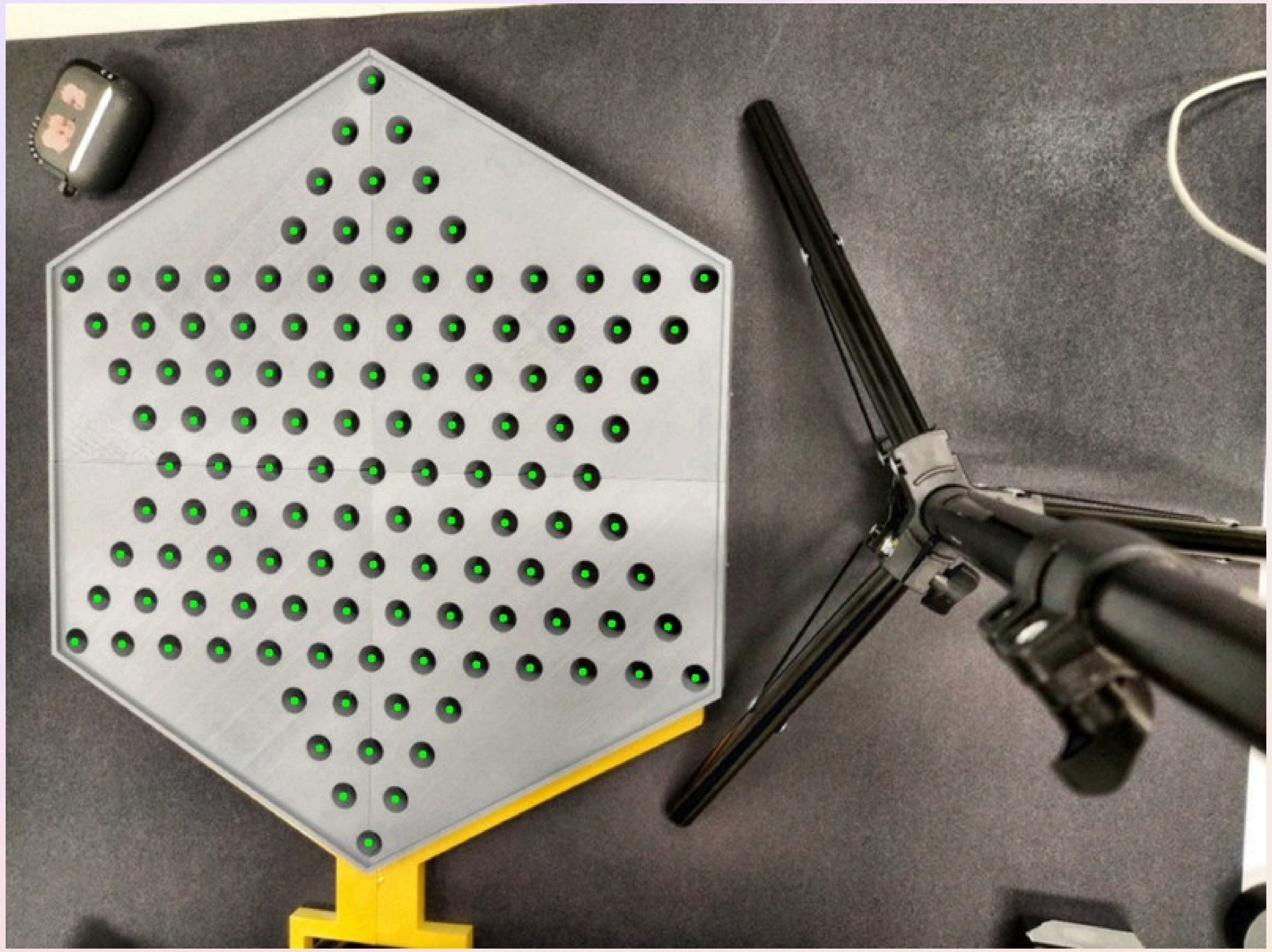
Counter detection
- To retrieve outermost contours and isolate the board's perimeter.

2.2

Cell Detection

Computer Vision - Cell *Detection*

Overall Processes



Hough Circle Transform

- To identifies all circular shapes that could represent board cells

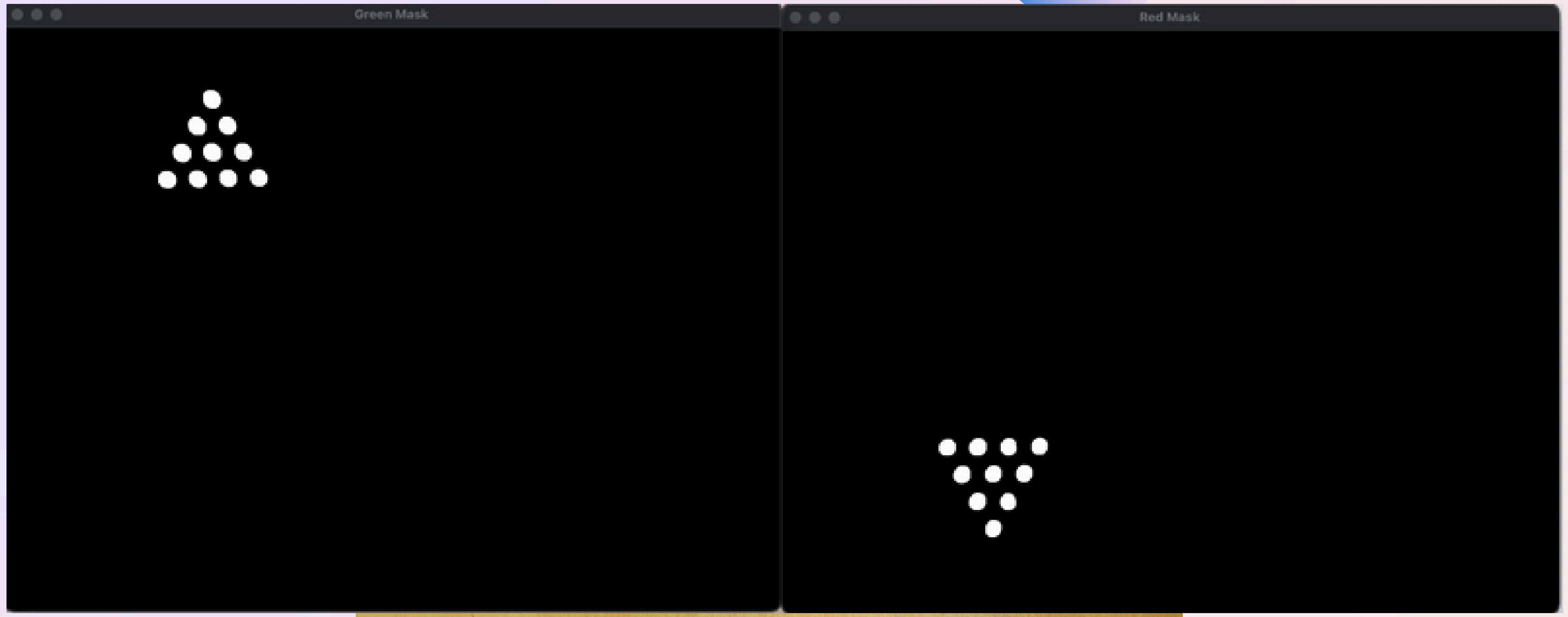
Board Contour Filtering

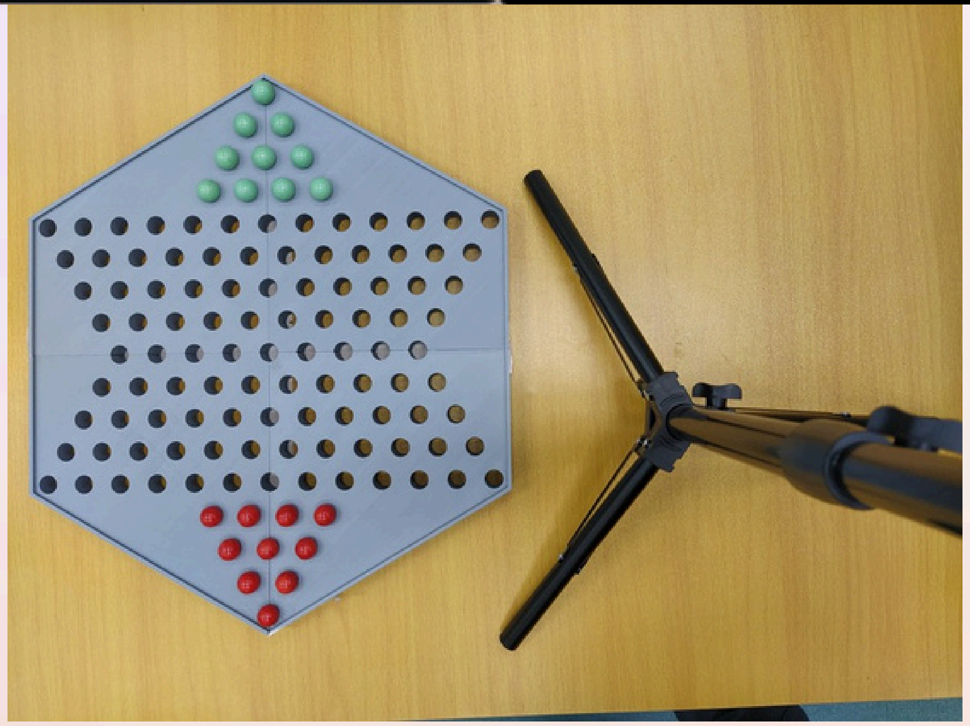
- To eliminate false positives outside the playable area

2.3

Marble Detection

Computer Vision - *Marble Detection*





Find Marbles

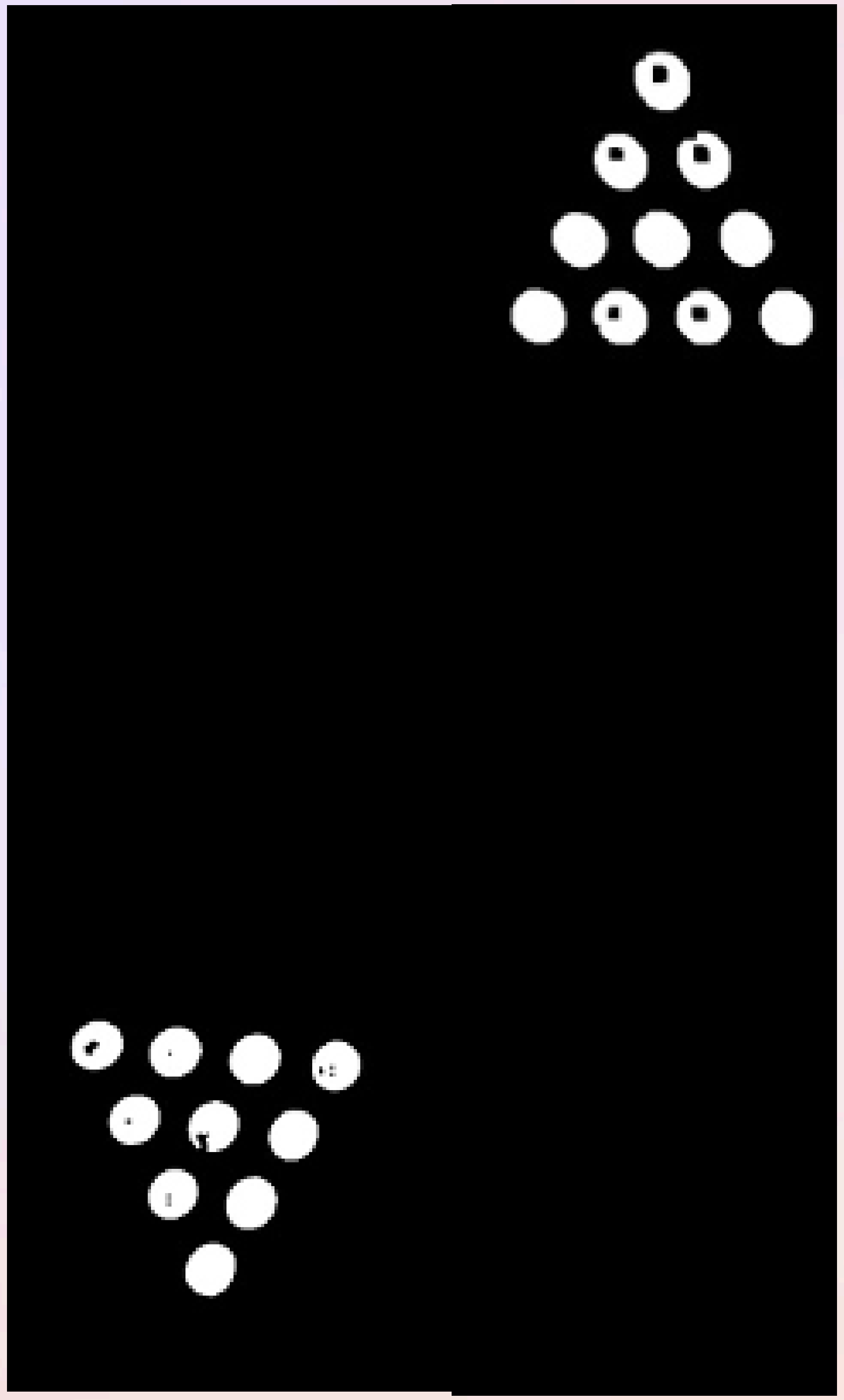
Identifies Colours (Red & Green) in the image

Morphological cleaning

Circle Detection from Masks

- Radius < 30
- circularity > 0.6: $4\pi \times (\text{area} / \text{perimeter}^2) > 0.6$
- Lies inside board

Marble Detection - Updates



Find Marbles

Identifies Colours (Red & Green) in the image

Morphological cleaning

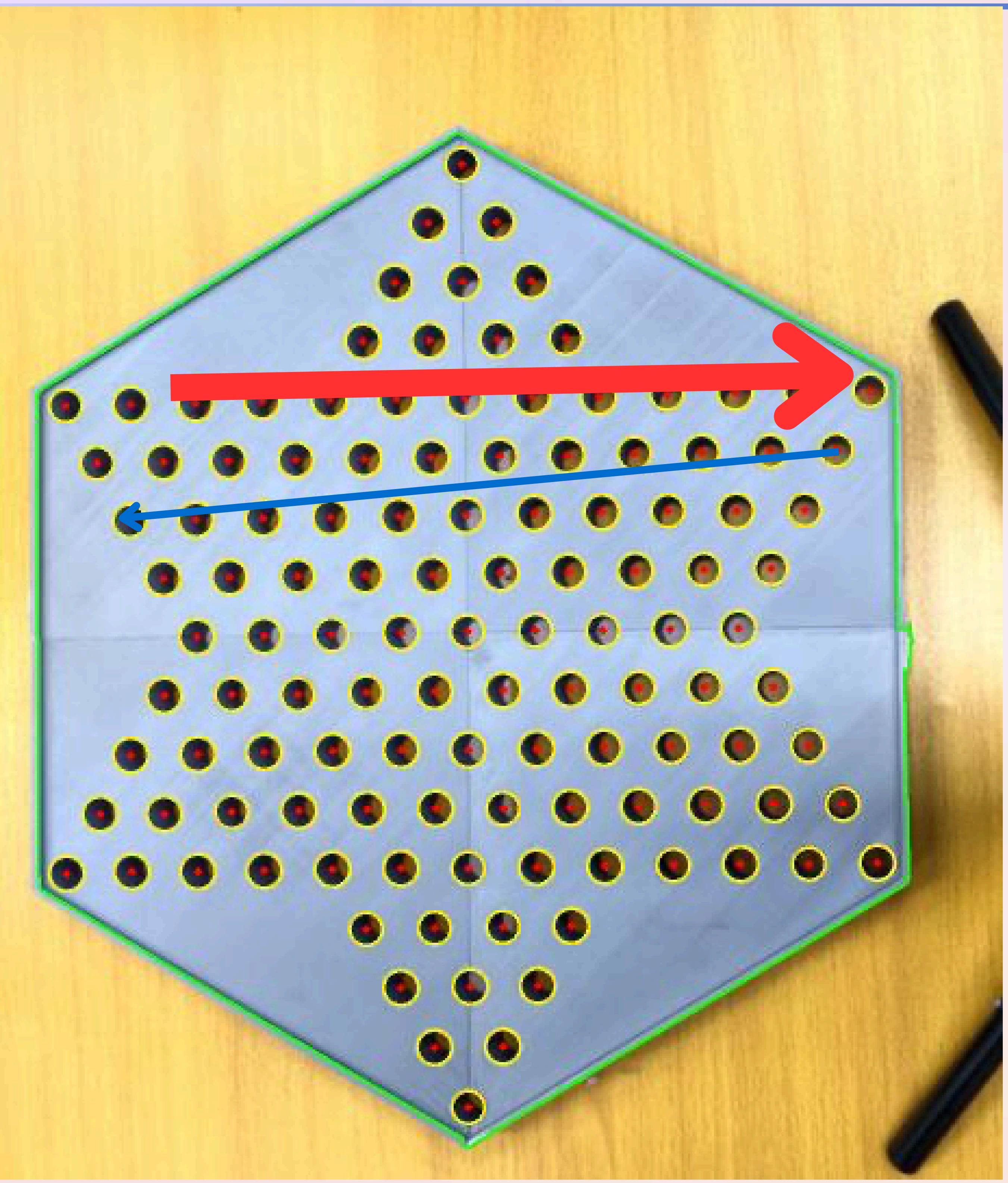
Ratio-of-Color (ROC) Detection:

- Radius 20px around each cells
- Classification threshold:
  - 15% of specific color = marble present
- Detection under uneven illumination, sunlight

2.4

Board State

Cells Assignment



Assign Cells to Array

Group cells (x, y) into rows if their y-coordinates are within 'row_threshold' of each other.

1. Sort by ascending y (then x as a tiebreaker).

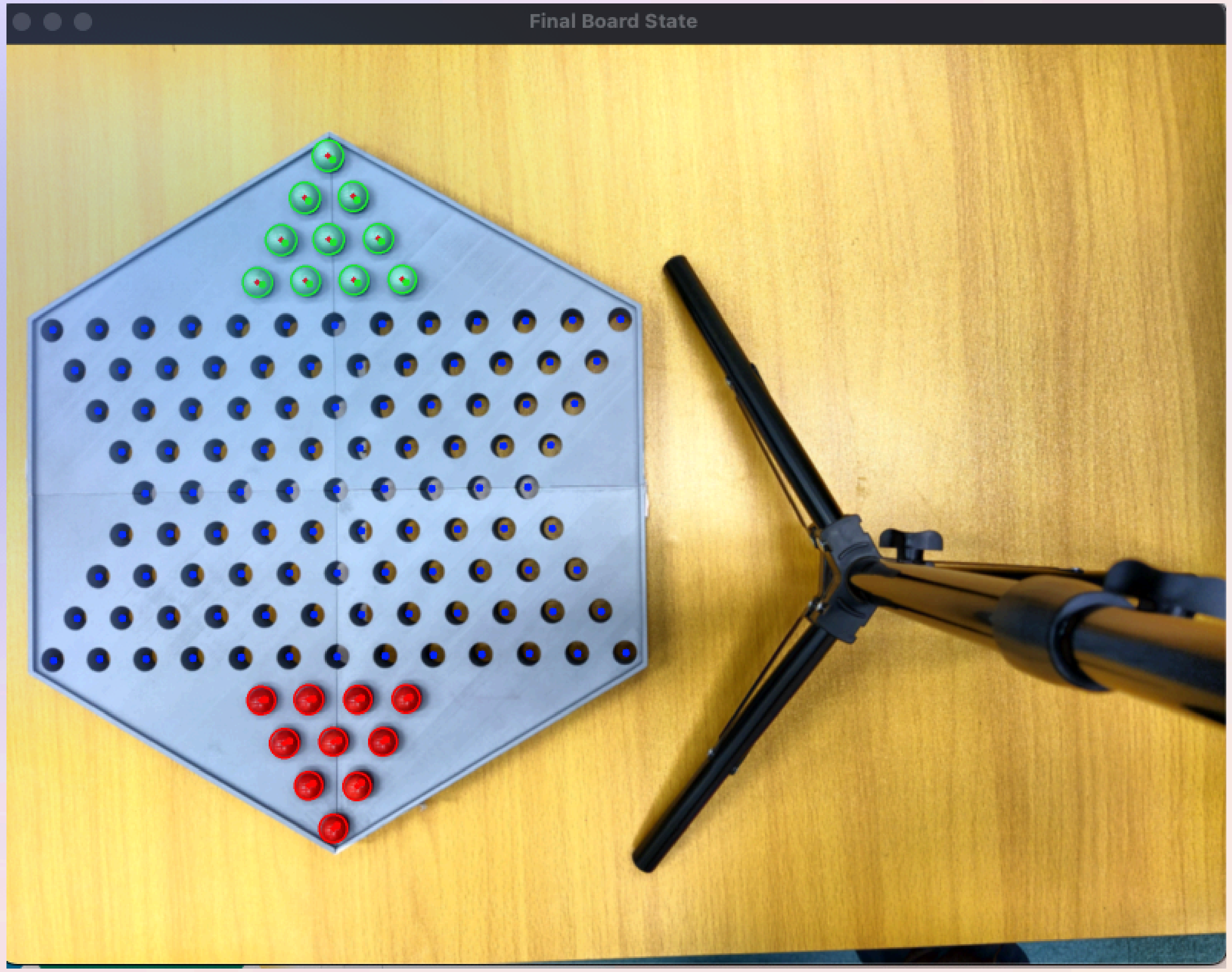
2. Start a new row when the y-difference is bigger than 'row_threshold', ~15

3. Sort each row by x ascending.

4. Return the grouped cells (flattened back into a single list, but now row-by-row).

```
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],  # row 4 has space for 13 cells
    [None, None, None, None, None, None, None, None, None, None, None, None],  # row 5 has space for 12 cells
    [None, None, None, None, None, None, None, None, None, None, None],          # row 6 has space for 11 cells
    [None, None, None, None, None, None, None, None, None, None],                # row 7 has space for 10 cells
    [None, None, None, None, None, None, None, None, None],                      # row 8 has space for 9 cells
    [None, None, None, None, None, None, None, None, None, None],                # row 9 has space for 10 cells
    [None, None, None, None, None, None, None, None, None, None, None],          # row 10 has space for 11 cells
    [None, None, None, None, None, None, None, None, None, None, None, None],    # row 11 has space for 12 cells
    [None, None, None, None, None, None, None, None, None, None, None, None, None],  # row 12 has space for 13 cells
    [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
]
```

Marbles Assignment



Assign Cells to Array

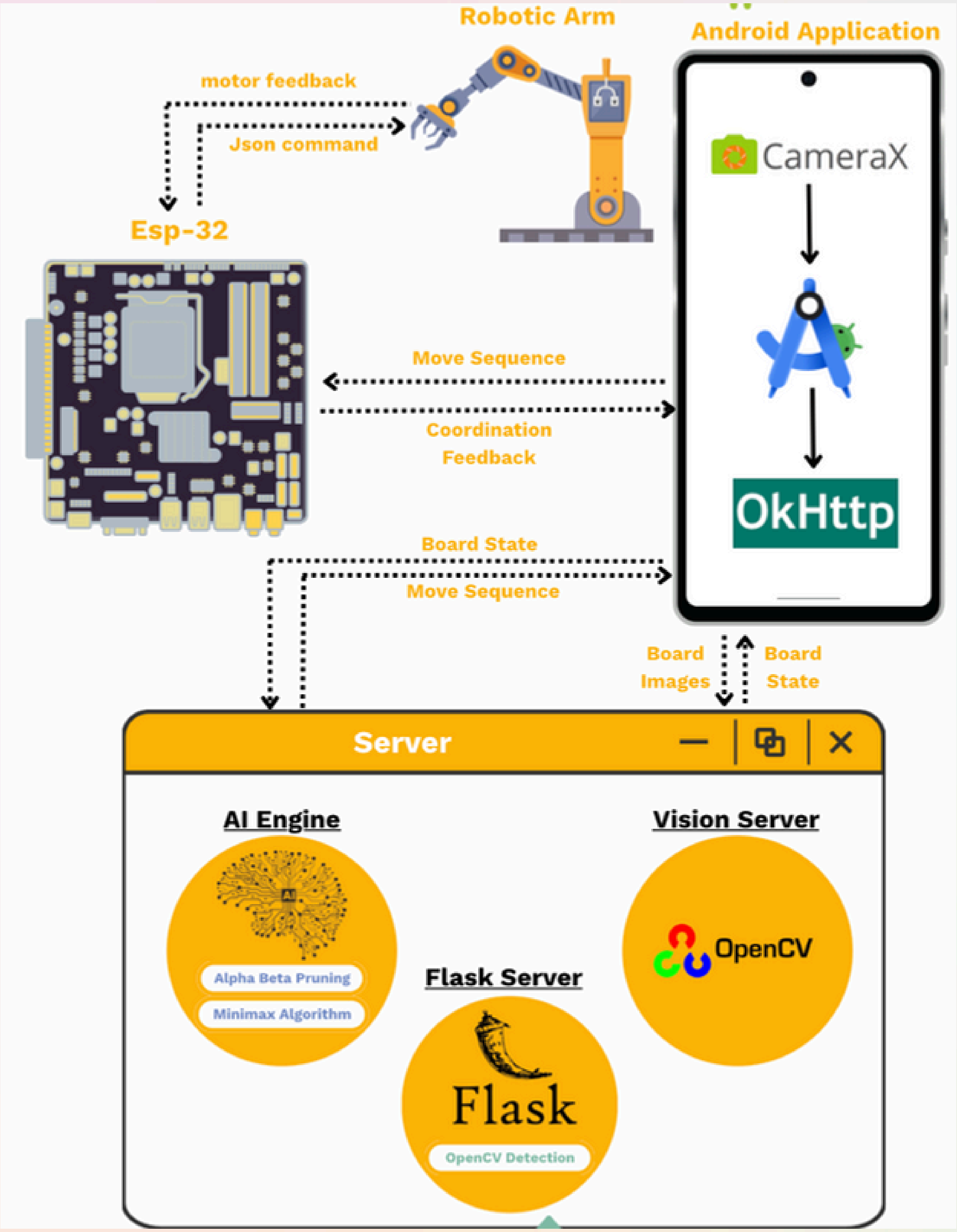
- Assign the marbles to the nearest cell
- Using Euclidean distance to sort

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

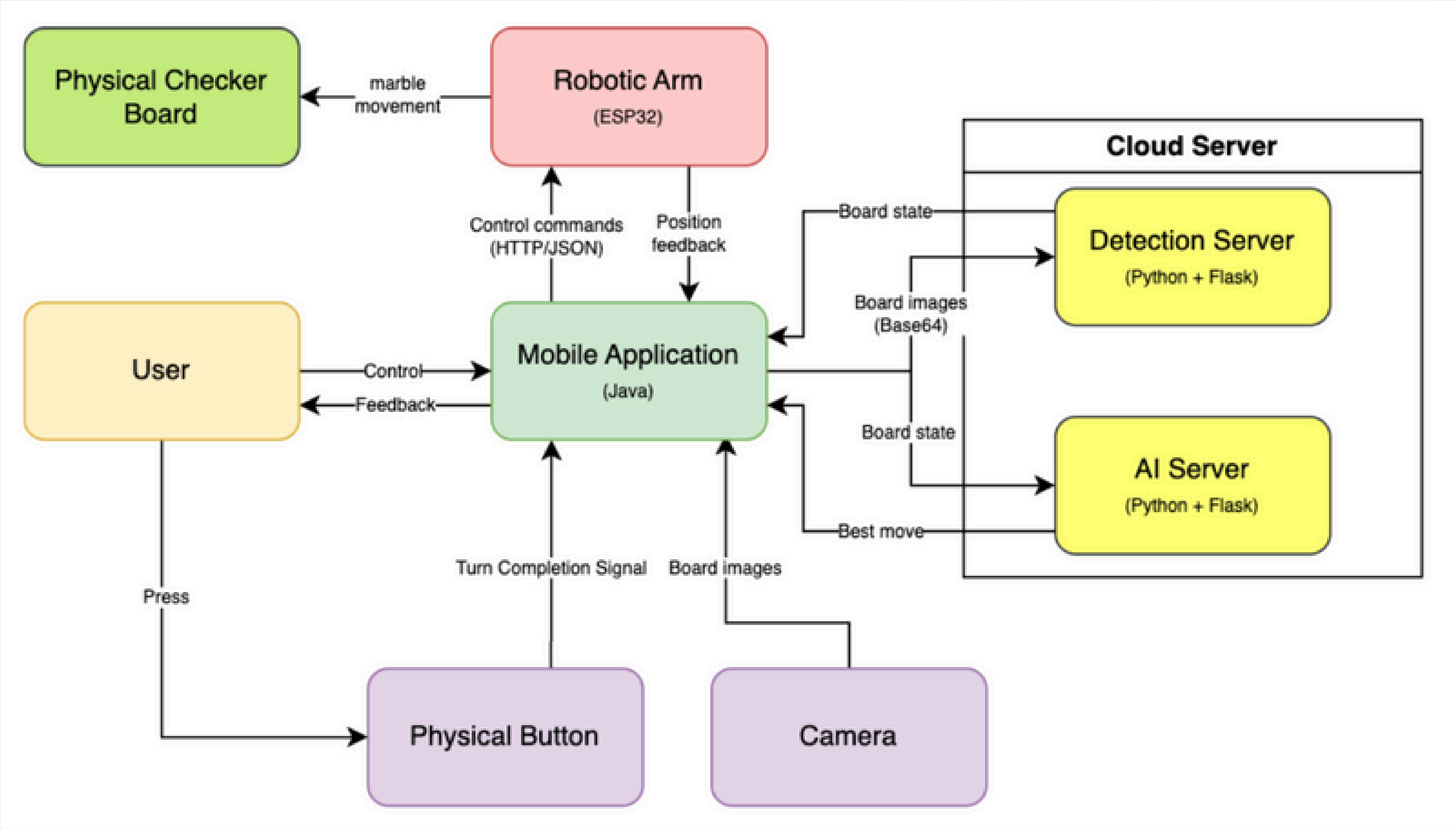
2.5

Communication Architecture

Overall System Architecture



Data Flow Of Our System



Cloud Server - Render

Render

1. **Static IP**
2. **<1s** response time
3. Multiple Server Location
4. **CI / CD** pipeline

03

AI Server

MiniMax Algorithm with Alpha Beta Pruning

Theoretical Foundation

- **Terminal state** (e.g., goal achieved or game over):

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

- **MAX's** turn: Our turns:

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

- **MIN's** turn: Opponent's turns:

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

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

Where:

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

Where:

- $\mathcal{M}(s)$ is the set of all successor states reachable from state s
- eval(s) is the evaluation function

Game Tree Structure and Analysis

```
                 Root (Initial State)
               /       |        \
     Move1   /         |         \   MoveN
           /           |          \
        State1      State2  ...  StateN    [Depth 1]
       /  |  \      /  |  \     /  |  \
      /   |   \    /   |   \   /   |   \
   States from opponent's responses    [Depth 2]
      /        |         \
     /         |          \
 States from AI's next moves          [Depth 3]
```

	Typical Mid-Game Value	Rationale
Branching factor	25 – 35 legal paths per ply	Each piece has ≈ 2–3 adjacent steps plus a handful of multi-jump paths; ten pieces with mutually exclusive destinations give high-20s branching.
Search depth	3 or 4	Depth 3 ≈ one full move by each player + the reply half-move; depth 4 adds the opponent's full reply.
$N \approx b^d$	b=30 ⇒ N(3)=27,000; N(4)=810,000	Flat-tree count before pruning or transposition savings.

Alpha Beta Pruning Enhancement

Pruning maintains two values

- **α**: The best value MAX has found so far
- **β**: The best value MIN has found so far

During Search, IF:

At a MAX node, the current value ≥ β,
search of this node's remaining children is skipped

At a MIN node, the current value ≤ α,
search of this node's remaining children is skipped

3.1

Evaluation Functions

Distance-Based Progression

Tile Distance Scoring System

Give every tile a "distance score".

The farest tile right inside your goal triangle is worth **16 points**.

One step farther away is worth **15**, the next **14**, ... until the farthest possible tile, which is worth 0.

In addition, if a tile belongs to the goal triangle of a colour, it gets an **extra bonus of +5** for pieces of that colour.

For any piece on tile t:

$$\text{score}_1(t) = (16 - d_1(t)) + 5 * \mathbf{1}_{\text{goal}}(t)$$

Where:

$\mathbf{1}_{goal}(t) = 1$

if t **already lies inside that side's goal triangle**, 0 otherwise

$D_1(t)$

shortest hop-count (in tiles) from tile t to the deepest corner of Player 1's goal triangle

Key Benefits:

- Creates a smooth gradient toward goal
- Incentivizes reaching the goal triangle (+5 bonus)
- Pre-calculated distances make evaluation efficient

Board Control/ Central - Lane Progress

Evaluation Function 2 rewards marbles for three things:

(1) **vertical progress**—every row it advances toward its home triangle is worth 10 points;

(2) **central alignment**—each column it drifts away from the row's centre line subtracts 1 point, encouraging pieces to stay in the quick "central lane";

(3) **goal completion**—the moment a marble enters its destination triangle it receives a flat +50 bonus, making arrival vastly more valuable than any single step elsewhere.

For any piece on tile t_i:

$$s(t_i) = \underbrace{10\boldsymbol{v}(t_i)}_{\text{vertical progress}} - \underbrace{\delta(t_i)}_{\text{side-step penalty}} \underbrace{+50 * \mathbf{1}_{\text{goal}}(t_i)}_{\text{goal-triangle bonus}}$$

Where:

Vertical progress $v(t_i)$

- number of rows the marble has advanced **towards its own destination triangle**.
- For the upward-moving side this is $16 - r(t_i)$; for the downward-moving side it is $r(t_i)$

Side-step penalty $\delta(t_i)$

- $\delta(t_i) = \text{abs}\,(x(t_i) - c(r(t_i)))$ = horizontal distance (in columns) from the board's centre line in that row.

Goal-triangle indicator

$$\mathbf{1}_{\text{goal}}(t_i) = \begin{cases} 1 & \text{if } t_i \text{ lies inside the player's target triangle} \\ 0 & \text{otherwise} \end{cases}$$

Heuristic Strategies for Chinese Checkers

Forward Directional Heuristic

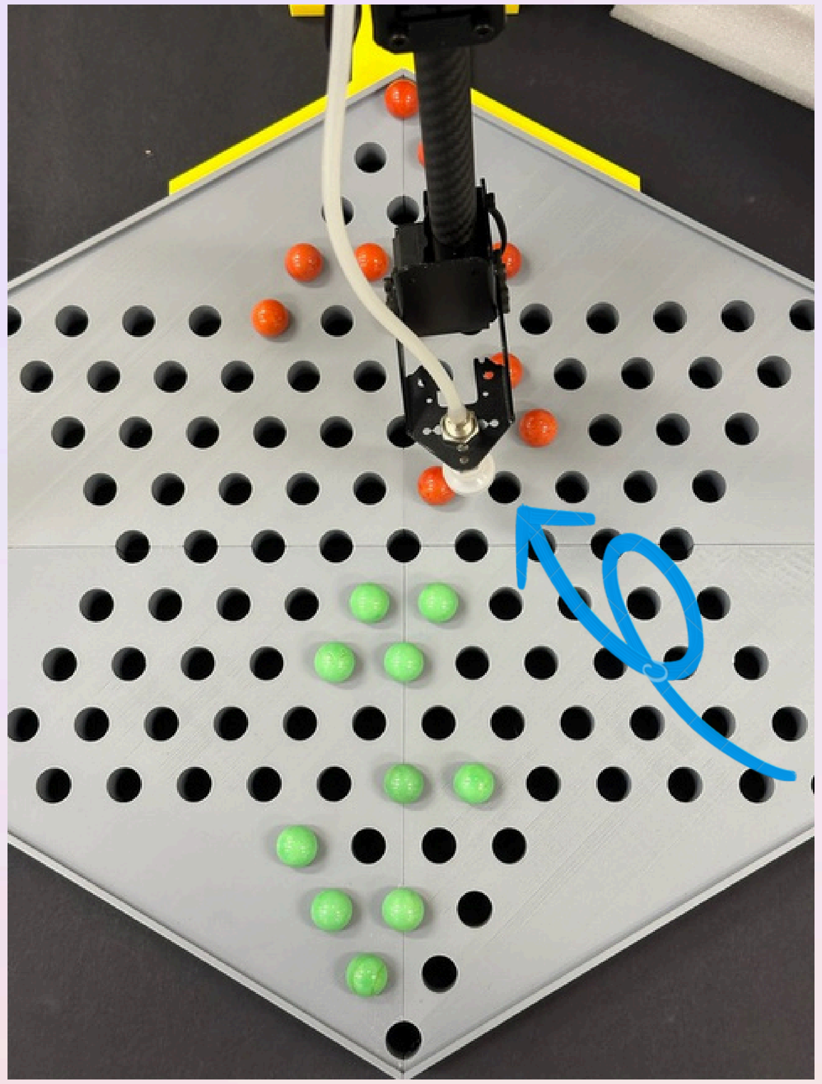
```
def heuristic(tile_origin, tile_destination) -> bool:
                # Enforce directional movement based on player's position
                if is_player1:
                    return              self.board.get_row_index(tile_destination)              >=
self.board.get_row_index(tile_origin)
                else:
                    return              self.board.get_row_index(tile_destination)              <=
self.board.get_row_index(tile_origin)
```

04

Robot Arm

Robotic Movement Correction

Servo Positioning Limitations



Mechanical Backlash: Play in the gears and joints causes positional discrepancies when approaching the same coordinates from different directions

Torque Inconsistency: The servos experience varying loads depending on the arm's extended position, leading to position drift under different weight distributions

Servo Resolution Limits: The 12-bit encoders provide theoretical 0.088° positioning resolution, but mechanical factors reduce actual precision. It is actually far from achieving this number.

Robotic Movement Correction

Strartegy Details

Adaptive Overshoot Strategy

When position errors exceed the acceptable threshold, our correction algorithm employs a progressive overshoot approach:

1. **First Attempt**: Standard movement to target coordinates
2. **Position Verification**: Measurement of actual position achieved
3. **Error Calculation**: Computation of X-Y ***positional error*** from target
4. **First Correction**: Command position at (target + error), ***essentially doubling the movement vector*** to compensate for systematic undershoot.
5. **Second Verification**: Re-measurement of position after correction
6. **Subsequent Corrections**: If still outside tolerance, apply overshoot factors (up to 3 total attempts), namely measure the difference of the actual coordinate now, comparing the original coordinate.

Feedback-Based Position Correction Algo

First Attempt

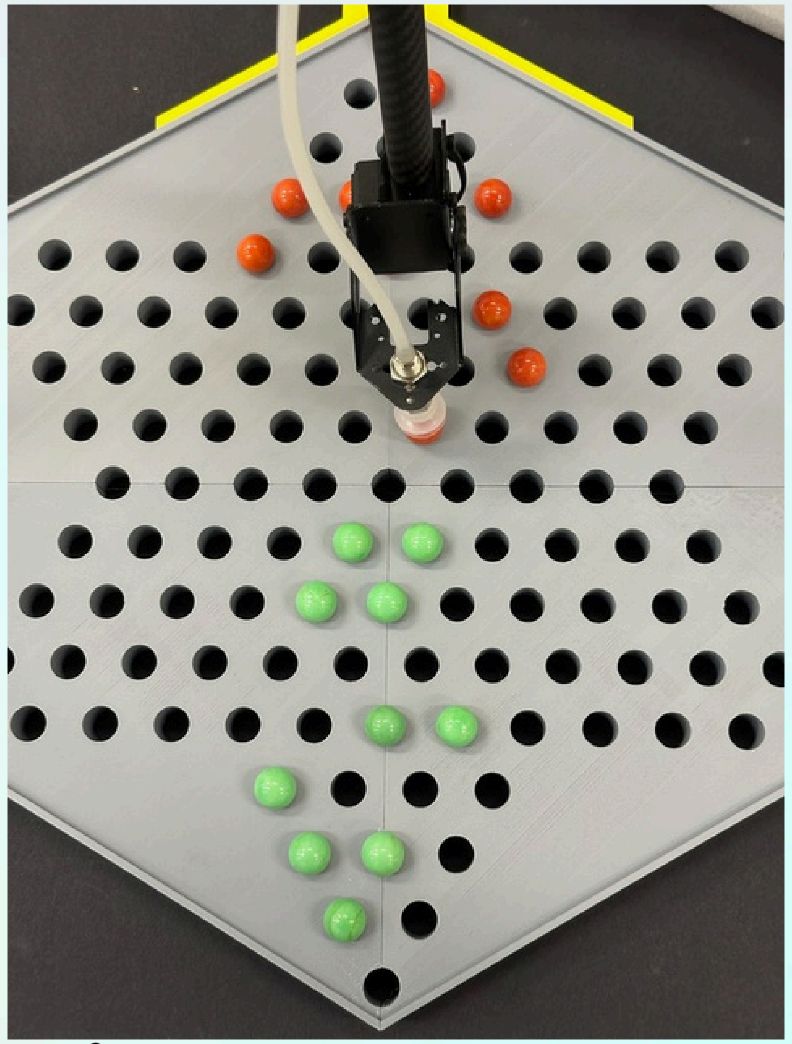
Position Verification

Error Calculation

First Correction

Second Verification

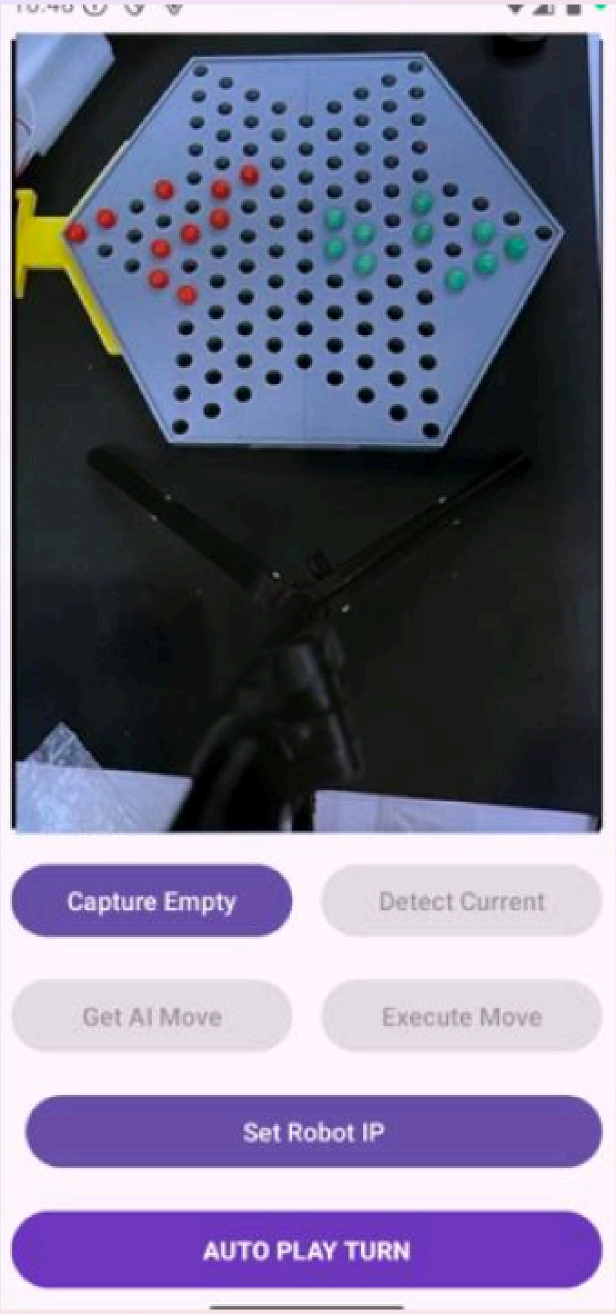
Subsequent Corrections



05

Mobile App

UI/UX Design



Top part

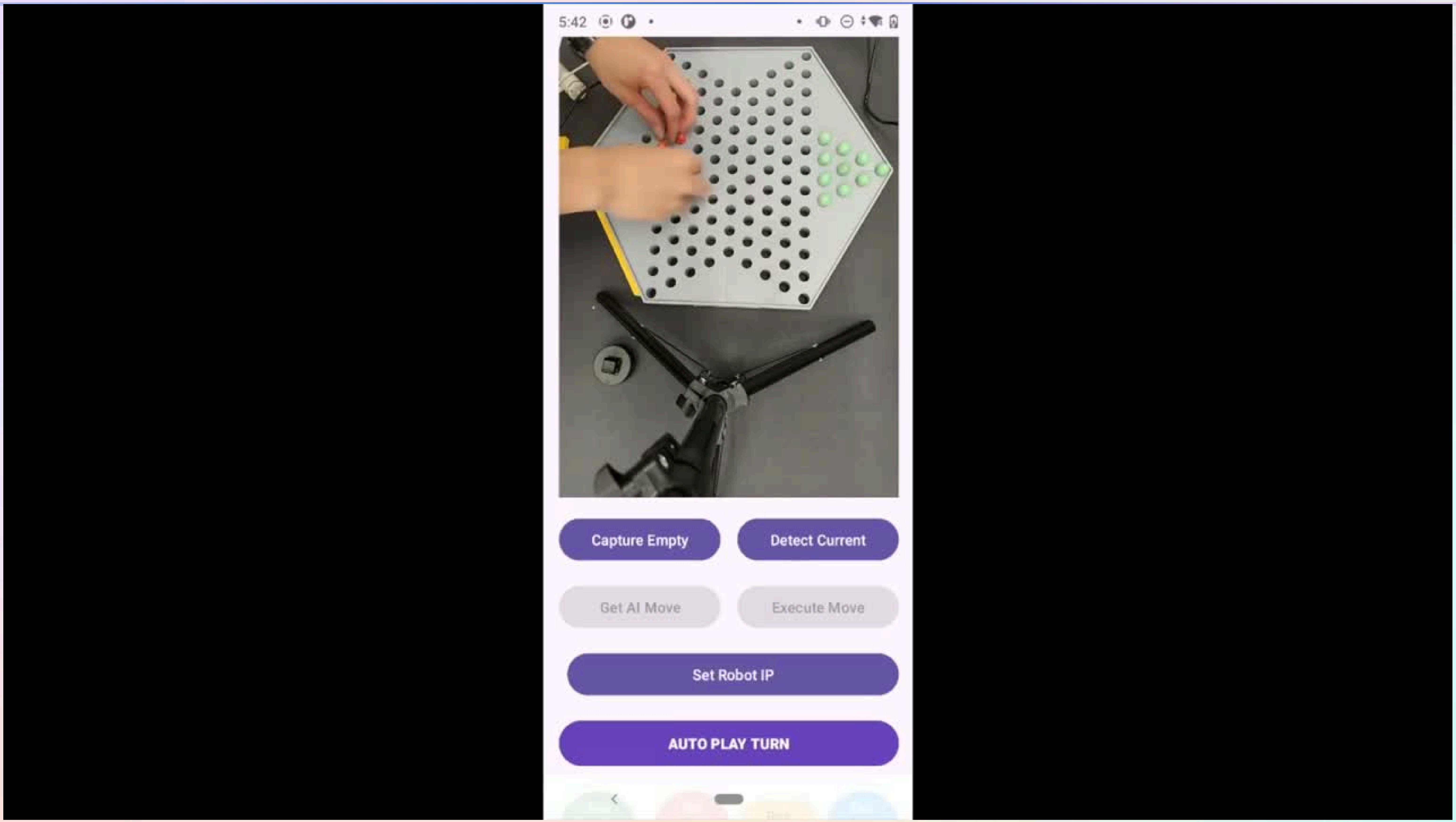
Bottom part

Runtime Screen Example

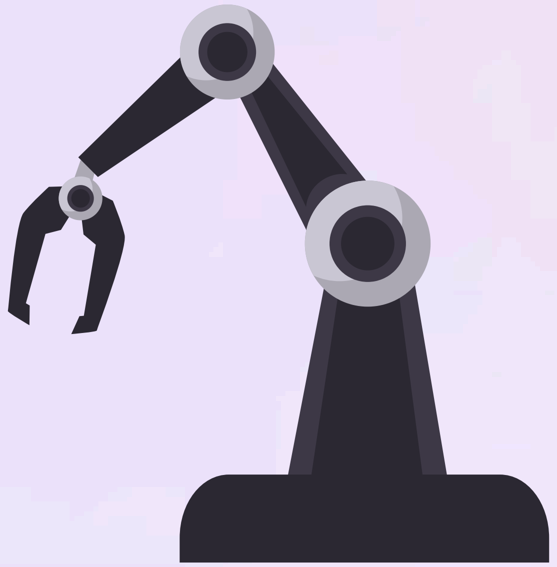
06

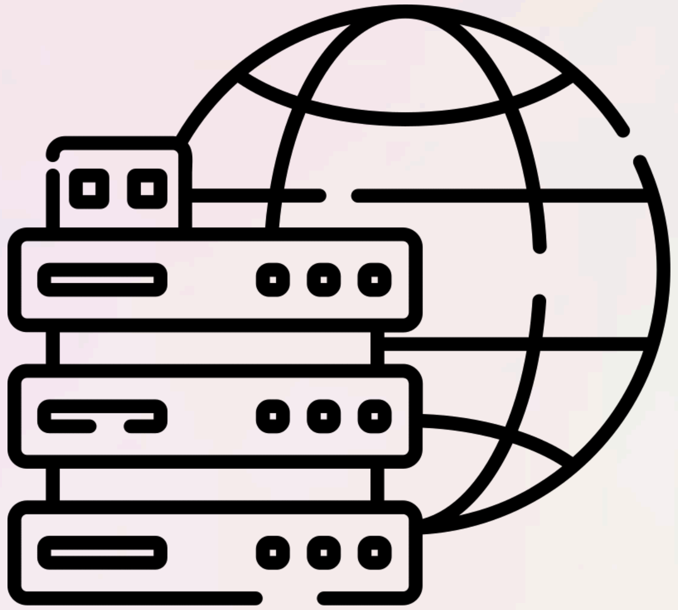
Quick Demo

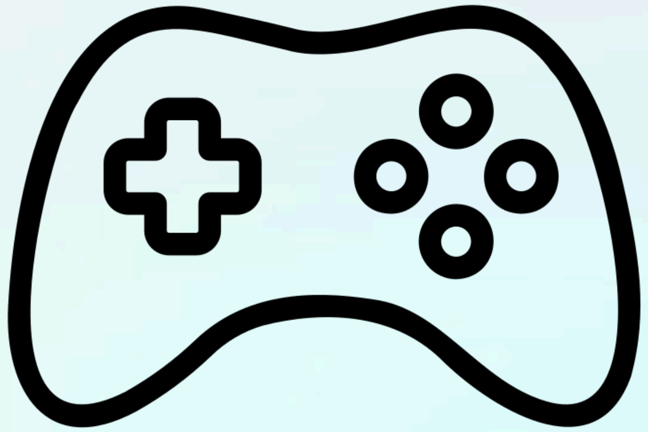
Demo Video


5:42
Capture Empty
Detect Current
Get AI Move
Execute Move
Set Robot IP
AUTO PLAY TURN

Future Work







- Hardware Enhancement
- Detection & AI Server Enhancement
- Additional Gameplay features

Perfecting the Robotic Arm

High-precision Servo Motors

- **Higher** Torque capability
- **Finer** Encoder resolution

- Greater positional **accuracy**, reduced latency, smoother movement trajectories

Optimized End Effector

- Integrated proximity sensors for adaptive height control
- Force sensors to optimize grip pressure for different marbles

- Significantly reduce average movement duration
- Create a more **fluid and engaging** experience

Expected Outcome:

- Execution time reduction to: **<4 seconds per jump**
- First-attempt success rate improvement for piece manipulation
- Better user experience closer to **human play speeds**

Detection and AI Server Improvement

Increase Server Coverage

Set up **multiple new servers** in different locations around the globe (currently in Singapore)

Multiple Gaming Sessions

Allow **multiple player instances** (different players can access the server at the same time)

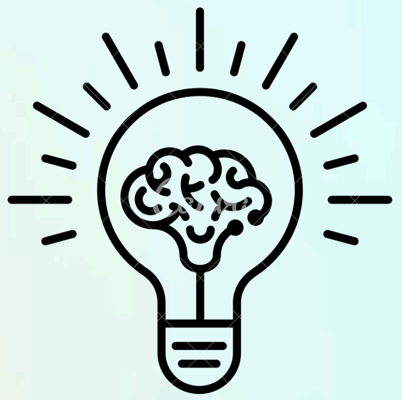
Commercialize our whole system into a gaming product that can sell to the public

Increase Computational Capacity

- **Multi-threaded search** for faster parallel evaluation
- **GPU acceleration** for heuristic scoring and deep tree exploration

Neural Network Evaluation

Train CNNs or Transformers on simulated or expert-level gameplay data.

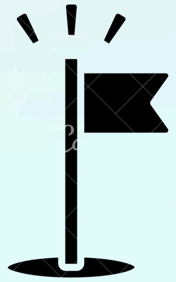


Make the AI smarter and have **higher search depth** (from 3 plies to **9 plies**)

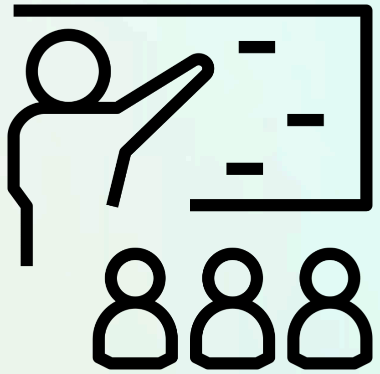
Additional Gameplay Features

Enriching user’s gaming experience

- **Multi-player support**: Extending AI logic to handle 3-6 player configuration
- **Game recording and replay**: Enabling review, analysis, and learning
- **Interactive tutorial mode**: Guiding users through strategies or robotic operations



- **Broaden** the platform's appeal across **educational**, casual, and **competitive** contexts



Autonomous Chinese Checkers Playing Robot Arm

Thank You.
Thank You.
Thank You.