[MakerLab] Pick and Place Game with Mobile App using Robotic Arm (Group 2)

Final Report

Chung Pui Yin 3035569160

u3556916@hku.hk

Supervisors: Dr. T.W. Chim and Mr. David Lee

Date of Submission: 18-4-2022
Acknowledgement

During the year of completing this project, I had received help and suggestions from several people. Without them, the progress would not go smooth. I would like to express my gratitude towards those who had helped me to finish this project.

First of all, I would like to thank Dr. Chim Tat Wing and Dr. Chui Chun Kit for providing suggestions about the project. After the interim presentation, they provided different opinions and suggestions on how to improve the app and inspired me to work out solutions to solve the problems.

Secondly, I would like to thank Mr. David Lee who is the technical staff. He provided many resources for me to make an android app and helped me to solve problems regarding the robotic arm. During the development of the app, problems regarding electronic board was encountered. He helped me to check the problem and found that the electronic board was not working. Finally, he solved the problem for me so that I can continue to test with the robotic arm.

Finally, I would like to thank my family and all those supported me. In March, I was infected by Omicron virus. It was a hard time for me to keep up with the project and other assignments. Thanks to my parents and relatives, they took good care of me and helped me to recover quickly by giving all the best to me including food and all the support. Without them, the progress of the project will be slowed down a lot and I would not be able to recover so fast.
Abstract

This project aims to make a mobile app controlling a robotic arm to play Othello Chess with a human player. Although there are many different robotic arms used in the industry including product packaging and sports, this project aims to make an interacting mobile app to control a robotic arm for entertainment.

The program gets the chess board information through computer vision using OpenCV for determining the objects captured from the camera. The program sends signal to control the robotic arm through Bluetooth and Arduino board which takes input in the form of G-Code and returns output to control the robotic arm to perform actions.

Currently, manual control and automatic control are completed. The app can extract game information from the camera to determine the objects.

Limitations and difficulties are encountered, the detection method can work but it is sensitive to the environment. Therefore, other detection method may have a better performance. Also, the robotic arm seems to be a bit damaged so the final position that the robotic arm arrives may be affected a little bit.

For future works or improvements, the design of the flipper, method for preparing new chess for the robotic arm, detection method and accuracy of the robotic arm can be improved.

As a result, under optimal environment setup, the app and robotic arm functions normally to play an Othello chess game automatically.
### Table of Contents

**Acknowledgement**

**Abstract**

**List of Figures**

1. **Introduction**
   1.1 Robotic Arm and Mobile App
   1.2 Objective and Deliverables
   1.3 Report Outline

2. **Methodology**
   2.1 Mobile App
      2.1.1 Chessboard Update
      2.1.2 Game Event Detection
      2.1.3 Getting Placeable Locations
      2.1.4 Getting Chess Locations to be Flipped
      2.1.5 Placing and Flipping Chess
      2.1.6 No Available Moves
   2.2 Computer Vision
      2.2.1 Chessboard Detection
      2.2.2 Chess Detection
      2.2.3 Hough Lines Transform
      2.2.4 Hough Circles
      2.2.5 Canny Edge
      2.2.6 Cluster
      2.2.7 Intersections
   2.3 Robotic Arm
      2.3.1 Steppers
      2.3.2 Gripper
   2.4 Arduino

3. **Results and Discussions**
   3.1 Results
      3.1.1 Manual Controls
      3.1.2 Automatic Controls
      3.1.3 Tests
   3.2 Limitations and Difficulties Encountered
      3.2.1 Failed Method
      3.2.2 Current Detection Method
      3.2.3 Flipping
      3.2.4 Slow Action
      3.2.5 Refill Chess
      3.2.6 Robotic Arm
      3.2.7 Bluetooth Connection
   3.3 Future Work / Improvements
      3.3.1 Improved Flipper
      3.3.2 Automatic Chess Provider
      3.3.3 Improved Detection Method
      3.3.4 Robotic Arm

4. **Conclusion**

5. **References**

6. **Attachment A**

7. **Attachment B**
List of Figures

- Figure 1: Placeable position calculation  
  P.11
- Figure 2: Chessboard grid line detection  
  P.13
- Figure 3.1: Captured image with white filter applied  
  P.14
- Figure 3.1: Chess detection  
  P.14
- Figure 4: Hough line representation  
  P.14
- Figure 5: Hough circle representation  
  P.15
- Figure 6.1: Edge gradient equation  
  P.16
- Figure 6.2: Gradient direction and edge  
  P.16
- Figure 6.3: Intensity gradient and threshold comparing  
  P.17
- Figure 7: Intersection of 2 lines representation  
  P.18
- Figure 7.2: Equations of $u_a$ and $u_b$  
  P.19
- Figure 8: Stepper  
  P.20
- Figure 9.1: Principle to pick up and put down object  
  P.21
- Figure 9.2: Comparison of the two types of grippers  
  P.21
- Figure 10.1: Initial state of Othello chess  
  P.23
- Figure 10.2: Available positions for white chess player (red outline)  
  P.23
- Figure 11.1: Comparing the chessboard with an image fails  
  P.25
- Figure 11.2 & 11.3: Failed detection detecting rectangles  
  P.25
- Figure 11.4: Reflection of light affects detection  
  P.26
- Figure 12.1: Lego Flipper  
  P.27
- Figure 12.2: Flipper top-down view  
  P.27
- Figure 12.3: Cross-section of the flipper  
  P.27
- Figure 13: Improved flipper design  
  P.29
- Figure 14: Automatic chess roll out device prototype  
  P.30
- Figure 15: Starting position of the robotic arm  
  P.38
- Figure 16: App user interface  
  P.39
1. Introduction

The following chapter introduces this final report. Firstly, a brief background of the robotic arm and mobile app is presented. Then, the objective and deliverable are introduced. Finally, the outline of the report is listed.

1.1 Robotic Arm and Mobile App

There are different kinds of robotic arm and robotic tools to help making jobs easier nowadays. However, there is not much on the field of entertainment for humans. Currently, most of these robotic components are designed to help with humans’ work like packaging products in factories, performing certain step in the progress of making a product. Also, there are some but not much on interacting and entertaining humans. For example, a robotic arm had been made to play table tennis with human which can detect the ping pong ball and hit back the ball like a normal player. However, these are just on the level of production and sports entertainment.

Due to the fast progression of technology in the modern world, smartphones and tablets are very common. Apps installed are designed to be convenient and able to finish all kinds of tasks with simple controls. A large variety of apps are available on the internet including games, tools, social platforms. However, there is not much on the linkage with robotic arm to interact with real-world objects. Also, a significant advantage of using app is that it is portable. For example, one can use the apps installed on the smartphone anywhere but desktop and laptop are too big and heavy for some situation and less portable.

In this project, the main product to be made is the mobile app that controls the robotic arm to act as a real player to play games with a human player. The app can get the real-world information through the smartphone camera and control the robotic arm to pick up and place down objects like chesses.
1.2 Objective and deliverables

In this project, the final deliverables are a robotic arm, a chess flipper and a mobile app for android phone. There are several functionalities that the robotic arm can perform which are movement, picking up an object and putting down an object. There also several functionalities the mobile app can perform which are capturing the game state, calculating where to place or flip the chess and coordinates for the robotic arm to move to, sending signals to the robotic arm and acting as a control panel for the user to control the robotic arm manually.

The robotic arm can be controlled manually or automatically by the mobile app to move along 3 axes of directions in the 3D plane (X, Y, Z) which is forward, backward, left, right, upward and downward. For manual control, the user can access the mobile app to control the robotic arm by tapping some virtual buttons in the app including moving the robotic arm, grabbing and releasing object. For automatic control, the phone must be placed at a place which can capture the game board and the game objects clearly to identify the game state. The app will send signal to the robotic arm to control its actions including moving to a position, grabbing an object and releasing an object. The two modes can be switched by tabbing the switch button in the app.

The mobile app contains several functionalities, including capturing the game state, calculating the available moves, sending signal to the robotic arm and acting as a control panel in manual control mode. Therefore, the device must be installed with a working camera. The camera will keep capturing the game board and game objects so that the app program has access to the current game state. Then, the program will calculate the available moves according to the rules of the Othello chess game which is pre-coded in the program.
Therefore, the game available for this mobile app and robotic arm is limited to Othello Chess. After the calculation finishes, the corresponding controls will be sent to the robotic arm to perform movement and pick and place actions. For manual control mode, the app will display a control panel which contains virtual buttons for the user to control the robotic arm directly.

1.3 Report Outline

After the introduction, Chapter 2 states how the mobile app interacts with the robotic arm and how the app obtain the game information to calculate the corresponding positions, followed by the introduction and explanation of Othello chess game. Chapter 3 states the results and discussion of the app and robotic arm including difficulties encountered and future work. Finally, Chapter 4 concludes the project.
2. Methodology

There are 3 main parts of this project which are the programming part of the mobile app (software), the robotic arm (hardware) and the game, Othello Chess.

2.1 Mobile app

For the programming part, the mobile app is coded using Android Studio in Java language. The app functionalities include connecting to the robotic arm, a manual control mode to control the robotic arm manually, automatic mode to capture the game state and control the robotic arm remotely. To control the robotic arm using the app, socket connection must be established first. Then, the mobile app can send a string type message in G-code format as command to control the robotic arm by writing to output stream to the socket.

Game Logic

The flow of the automatic mode is as follows:

2.1.1 Chessboard Update

For every frame captured, each grid state is checked and recorded with counters. At the tenth frame, if the counter of a grid is larger than 5, the app will update the index in the chessboard array list to that state. In short, the chessboard array list is updated every 10 frames.

After the chessboard array list updates, the app will check the chess number. There are 3 cases which is the number increases, has no change and decreases.

2.1.2 Game Event Detection

If it increases, it means that a player places a new chess on the chessboard. Therefore, the app will find which chesses are to be flipped. After obtaining the flip list, the app will control the robotic arm to flip those chesses.
If it has no change, the app will check if there are any placeable positions. If there are not, the app determines that the current turn player passes the turn so the turn player has switched. Otherwise, if there are and it is the player’s turn, it will do nothing and keep updating the chessboard state. If there are and it is the robotic arm’s turn, the app will choose a random index to place the new chess. There are many methods to calculate the state value of a game but this project aims to make the robotic arm acts like a human player so the “best” methods like minimax or expectimax are not used. If the app always makes the “best” moves, it is just a machine that follows some rules which is boring. Instead, choosing random index makes the game more interesting as the player cannot predict where the robotic arm will place its new chess.

If it decreases, it is because the robotic arm is flipping one of the chesses. In this case, the app will not update the chessboard.

Also, while the robotic arm is moving, the app will not update the chessboard as the robotic arm will affect the detection while it is moving between the camera and the chessboard.

2.1.3 Getting Placeable Locations

To get the placeable grid index, the app searches every opponent’s chess and checks if the surrounding 8 grids are placeable. The definition of “placeable” is that the grid is empty which no chess occupies that grid and after placing the chess there, there should be only opponent’s chesses between the placed chess and another chess owned by the player. Following these rules, for each opponent’s chess, the app will check within the direction from the placed chess to the opponent’s chess, is there a chess owned by the turn player and there are only opponent’s chesses in between as shown in Figure 1.
2.1.4 Getting Chess Locations to be Flipped

For finding the chess to be flipped, it is like the above method but this time only the newly placed chess is used as a point to check the eight directions around it. If there is turn player chess at the end of a direction and there are only opponent’s chesses in between, then those opponent’s chesses indices are put into an array list. The list will then be passed to the flip function.

2.1.5 Placing and Flipping Chess

For placing and flipping chess, they work in a similar way where placing a chess only controls the robotic arm to move to the “new chess” position recorded and turn on the vacuum to pick up the chess and put it on the chessboard according to the input index. For flipping the chess, an array list is taken as input and the app controls the robotic arm to pick up chesses from the chessboard and put it in the flipper. Then, the app moves the robotic arm to the exit of the flipper to pick up the flipped chess and put it on the original position on the chessboard.

2.1.6 No Available Moves

A difference that makes Othello chess being special from other chess game is that sometimes a player has no legit moves to make so there is no choice but to pass the turn without placing any chess. The app can check how many
placeable position for the current turn player, the app switches turn player if it is 0. If both player and the robotic arm has no available moves, the game ends.
2.2 Computer Vision

2.2.1 Chessboard Detection

First, for the detection of the chessboard, the captured frame is filtered to detect the green area and the filtered image is in black and white where black represents area that is not in the green color range and white represents those in the green color range. Then, the filtered image is applied to the canny filter to make it easier to detect the lines. The lines of the chessboard grid are detected using Hough Lines detection provided by OpenCV. Next, the intersections of the horizontal lines and vertical lines are the corners of the chessboard grid. Finally, the corners are clustered into 81 corners of the 8x8 chessboard.

![Chessboard grid line detection](image)

2.2.2 Chess Detection

Next, for the detection of the white and black chess, similar to the chessboard detection, the captured image is applied to white color filter first. Then the filtered image is also black and white where white represents the white area and black represents those are not in white color range. The chess is in the shape of a circle so the white chess is detected using Hough Circles detection provided by OpenCV. For black chess, this time the original captured image is used and all circles are detected using the same method. The set of all circles
contains both white and black circles detected. Therefore, all of those that are not white circles are considered as black circles. Hence, the captured image should only contain the chessboard, white and black chesses.

![Figure 3.1: Captured image with white filter applied](image1)
![Figure 3.2: Chess detection](image2)

### 2.2.3 Hough Lines Transform

Hough line is a line represented in the form of $r = x \cos \Theta + y \sin \Theta$, where $r$ means the perpendicular distance from the origin to the line (also known as rho) and $\Theta$ is the inclined angle of rho and the $X$-axis. For horizontal lines, the theta is $0$ or $\pi$ and for vertical lines, the theta is $\pi/2$.

![Figure 4: Hough line representation](image3)

In this project, the lines to be detected are the grid lines which are either horizontal or vertical. Other inclined lines are ignored by filtering the lines with the theta value. Take vertical lines as example, vertical line has a theta value of
$\Theta = 0 \text{ or } \Theta = \pi$. Therefore, considering some error with the placement of the camera that makes some lines tilt a little, lines detected with theta value $\Theta = 0$ and $\Theta = \pi$ or $\Theta = \pi - 0.1$ are considered as vertical lines.

2.2.4 Hough Circles

For any circle, it can be constructed with 3 parameters $(x, y, r)$ where $x$ and $y$ represent the 2D coordinates pair $(x, y)$ of the center of the circle while $r$ represents the radius of the circle. Hough circle uses the rho mentioned in the previous subtopic as the radius but the rho is not the perpendicular distance from the origin anymore. Instead, the rho is the distance from the center of the circle to the circumference.

In this project, the chesses are in the shape of circle. Therefore, the center of the chess is used to calculate the grid index of the detected chess that it lies in.

![Figure 5: Hough circle representation](image)

2.2.5 canny Edge

Canny edge detection is an edge detection method which goes through different steps to get accurate detection. The method takes a grayscale image as input.
First, noise reduction is performed by removing the noise using Gaussian filter. Here, noise means the part that may affect the detection. This is to smoothen the image to reduce the misdetection of the method.

Next, the edited image is applied to a filter with a Sobel kernel in horizontal and vertical way to get the first derivative in respective directions, namely $G_x$ and $G_y$. By applying the values in the equation as follows:

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Figure 6.1: Edge gradient equations

The edge gradient and direction of a pixel can be obtained.

Then, for every pixel on the image, the pixel is compared with its neighbours to see if it is a local maximum. For example, in Figure 6.2, point A lies on the vertical edge and the gradient direction is perpendicular to the edge so point A is checked with point B and C to see if point A is a local maximum. If so, the method moves on to the next step.

Figure 6.2: Gradient direction and edge

The last step is to decide which edge is really an edge and which are not by setting 2 threshold values, maximum value and minimum value. Since the
intensity gradient is obtained in the previous step, for edge with value greater than the maximum value, it is sure to be an edge. For edge with value lower than the minimum value, it is sure to be not an edge. For edge with value between the maximum and minimum value, if that edge connects to a part that is sure to be an edge, then it is also considered as an edge but if not, that edge is considered as not an edge.

Figure 6.3 shows an example of edge detecting. Edge A has value greater than maximum threshold value so it is considered an edge. Edge C is between the 2 threshold values but it connects to edge A which is sure to be an edge so edge C is also considered as an edge. However, for edge B, it is neither greater than the maximum value nor connects to any edge. Therefore, edge B is not considered an edge.

2.2.6 Cluster

From the detection of horizontal and vertical lines, there are many lines detected which obviously exceeds the number of lines on a chessboard. Hence, the intersections formed by these lines also exceeds the number of corners on
a chessboard. Therefore, clustering is performed to get a representing point of each corner.

In this project, the clustering method uses the centroid of a group of points as the representing point. The app assumes the player to only capture the chessboard and chesses with the camera. Therefore, the intersections should only be the corners of a chessboard. The app considers points within a certain range as a cluster. The range is less than a grid length while it is large enough to include the corresponding points.

2.2.7 Intersection of Two Lines

The most common method of calculating a point of a line is using the line equation \( y = mx + c \) where \( m \) is the slope of the line and \( c \) is the intersection of the line with Y-axis. Calculating the intersection of 2 lines is to find a point with 2 line equations. However, in the situation of detecting the grid line on a chessboard, vertical lines may have an infinite slope or an extremely large slope and horizontal lines may have a 0 value slope. Therefore, using the simple line equation \( y = mx + c \) is not suitable.

Instead, with 2 points \( P1(x_1, y_1) \) and \( P2(x_2, y_2) \) on a line and another 2 points \( P3(x_3, y_3) \) and \( P4(x_4, y_4) \) on another line, the intersection can be calculated while avoiding the slope problem.

![Figure 7.1: Intersection of 2 lines representation](image)
The line equations are:

\[ P_a = P1 + u_a (P2 - P1) \text{ and } P_b = P3 + u_b (P4 - P3) \]

By solving \( P_a = P_b \), \( u_a \) and \( u_b \) can be simplified as:

\[
\begin{align*}
  u_a &= \frac{(x4 - x3)(y1 - y3) - (y4 - y3)(x1 - x3)}{(y4 - y3)(x2 - x1) - (x4 - x3)(y2 - y1)}, \\
  u_b &= \frac{(x2 - x1)(y1 - y3) - (y2 - y1)(x1 - x3)}{(y4 - y3)(x2 - x1) - (x4 - x3)(y2 - y1)}.
\end{align*}
\]

Figure 7.2: equations of \( u_a \) and \( u_b \)

Finally, the \( u_a \) and \( u_b \) can be substituted into the following line equation to get the intersection coordinates:

\[
\begin{align*}
  x &= x1 + u_a (x2 - x1) \text{ and } y = y1 + u_a (y2 - y1).
\end{align*}
\]
2.3 Robotic Arm

For the robotic arm, it is 3D printed and assembled with other mechanic parts like stepping motors. It picks up objects by using air pressure but not friction. Since the robotic arm is not made by me, the details can be found from the report by a previous student Mr. Yunhai Wang.

2.3.1 Steppers

The steppers / stepping motors used for the robotic arm in this project is of the type NEMA17. They can rotate with high precision to control the robotic arm to move in X, Y, Z directions.

![Stepper](image)

Figure 8: Stepper

2.3.2 Gripper

There are many kinds of gripper which can pick up and put down objects including claw gripper and vacuum gripper. In this project, the vacuum gripper is used. A comparison of the claw type and vacuum type gripper can be found in figure 9.2.

When the gripper of the robotic arm which is connected to a tube is placed on the object, the vacuum turns on to draw out the air inside the tube. It results in the air inside the tube has lower pressure compared to the air outside the tube. Since high pressure flows to low pressure, the air outside tends to flow into the tube. Hence, the object is pressed against the tube.
moves upward, the object will be lifted. Reversely, when the vacuum turns off to balance the air pressure inside and outside the tube, the only force acting on the object is gravitational force. Therefore, the chess falls. The robotic arm receives commands from the mobile app through Arduino board to perform actions.

Figure 9.1: Principle to pick up and put down object

Figure 9.2: Comparison of the two types of grippers
2.4 Arduino

The key for communication of the app and robotic arm is the Arduino board. The robotic arm itself only performs action and Arduino board receives commands. The Arduino board is connected to a Bluetooth chip to allow connections to Bluetooth devices. The app connects to the server socket of Arduino board via Bluetooth. Then the socket listens to G-code command string messages from the app and return physical outputs like turning on the steppers to move the robotic arm or turning on the vacuum to pick up objects.
2.5 Game

Another important element of this project is the game. As stated in the project title, the mobile app controls the robotic arm to play a game with the player. This project focuses on playing the Othello chess. It is a turn-based game with one player playing against another player on an 8x8 grid size chessboard. Initially, there are 4 chesses on the chess board as shown in figure 10.1. The rule to place a chess in the player’s turn is that there must be at least one opponent’s chess placed between the turn player’s chesses. For example, figure 10.2 shows the possible locations (outlined in red) for placing the white chess. After placing the chess, the opponent’s chesses between the turn player’s chesses turns into the turn player’s chesses (flip them to change the colour). The game ends if both players cannot place any chess on the chess board, including the situations that the whole grid is filled. Since this game requires players to flip the chess after every move, it is a bit annoying for the players to do so and sometimes it is difficult for someone with big fingers to flip while not affecting other chess position. Therefore, robotic arm can solve this issue and these two would be a great combination.

![Figure 10.1: Initial state of Othello chess](image1)  ![Figure 10.2: Available positions for white chess player (red outline)](image2)
3. Results and Discussion

The following includes the results of this project and the limitations and the difficulties encountered.

3.1 Results

3.1.1 Manual Controls

For the app, the manual control is implemented with simple button interactions by Java coding to send signals to the robotic arm through Arduino. The details of how to control the robotic arm can be found at Attachment B.

3.1.2 Automatic Controls

The game state is captured using the smartphone camera. Then the image is analysed using OpenCV to help detecting and identifying various objects. It detects the chess board and identifies the location of the chess objects. To convert the 2D coordinates on the captured image to 3D coordinates that the robotic arm takes as input, the player is required to calibrate the robotic arm by moving it to several grid locations on the chessboard, position of the flipper and where to pick new chess. Then, the player presses the calibration buttons to record these positions and the app will be able to calculate the other grid positions and perform automatic actions normally.

3.1.3 Tests

Several tests had been conducted. From observation, the position of the end point is calculated quite accurately that it can pick up and put down the chesses without any problems most of the time. There may be several times that it places a chess crossing the grid line but still in an acceptable range.
3.2 Limitations and Difficulties Encountered

3.2.1 Failed Method

While implementing the detection method, matching the captured image with an image to determine the object was implemented at first. However, the app cannot detect the chessboard at all as shown in figure 11.1. Also, it will not be detected once the chesses are placed on the chessboard. Therefore, this method did not work.

![Figure 11.1: Comparing the chessboard with an image fails](image)

Another method tried was finding rectangles as each grid is a rectangle. However, the edge of the chessboard contains many small sharp bumps which caused a lot of misdetections as shown in figure 11.2 and 11.3.

![Figure 11.2 & 11.3: Failed detection detecting rectangles](image)
The other detection method which is detecting the colors and shapes are used instead. Although it can work as intended under optimal environment setup, it is sensitive to the light and environment changes.

3.2.2 Current Detection Method

The current method is to use color and shape to determine the objects. However, this method is very sensitive to color. For example, the reflection of light will affect the detection as shown in figure 11.4 including not detecting the lines and corners or misdetection of the reflection of light as a white chess. Also, the reflection of light on a black chess will make the app think that it is a white chess. Moreover, if the lighting is not bright enough or the shadow on a white chess will also make the app thinks that it is a black chess. Therefore, the environment of detection must be set optimally to play the game.

![Reflection of light affects detection](image)

**Figure 11.4: Reflection of light affects detection**

3.2.3 Flipping

While playing Othello chess, players always need to flip the chesses after placing a chess. For the current robotic arm, the gripper is not designed to flip an object since it is supposed to move the object without changing its orientation. Therefore, external tool is required without modifying the mechanics of the robotic arm. The flipper is shown in figure 12.1 and 12.2
while figure 12.3 shows how the flipper flip a chess. For this project, Lego blocks are used to build the flipper due to its simplicity and it is easy to test once it is built or modified. However, a disadvantage of using Lego blocks is that it is difficult to show how it is built.

Figure 12.1: Lego Flipper

Figure 12.2: Flipper top-down view

Figure 12.3: Cross-section of the flipper

However, the current design is not optimal. There is a small chance that the flipper cannot flip the chess as there is space for the chess to flip twice or to not flip as intended. In such scenario, the player may need to flip it manually.

3.2.4 Slow Action

Another limitation is that flipping is another frequently performed action in Othello chess game. However, the robotic arm cannot perform such action quickly so it takes quite a long period of time if there are many chesses are to be flipped.
3.2.5 Refill Chess

For the “new chess” position used by the robotic arm to pick up new chess in its own turn, the player needs to refill the new chess by manually placing a black chess at that position. An external device can be used for automatically refill the chess but there is no time to implement it in this project so it requires the player to refill it manually.

3.2.6 Robotic Arm

Concerning the hardware which is the robotic arm, it cannot move to a position accurately sometimes. According to observation, the gear at the bottom for rotating seems to get stripped. When the robotic arm moves to a position, one can assert so little force to move it along the X-axis for a short distance without controlling the robotic arm to move. However, the Othello chessboard is not that big and a small difference on the target position may cause the chess to place on another chess or cannot pick up a chess on the chessboard.

3.2.7 Bluetooth Connection

Lastly, the app only pairs to Bluetooth device with the corresponding address and UUID. One can change it in the source code.
3.3 Future Work / Improvements

As mentioned above, the flipper sometimes cannot flip the chess, the player needs to refill a new chess manually, the chess game objects detection is not accurate under certain situations and the robotic arm may moves to a wrong position.

3.3.1 Improved Flipper

To improve the flipper, the design can be changed to the following design in figure 13. This design does not leave extra space for the chess to over-flip or not flip so it ensures the chess can flip after passing the tunnel. It is better to make it with 3D printing as Lego blocks cannot tailor make the radius of the tunnel to fit for the chess to pass through.

![Figure 13: Improved flipper design](image)

3.3.2 Automatic Chess Provider

To refill the new chess automatically, a device can be made to let the chess roll out after the new chess is taken away. The design is that the player inserts all 64 chesses into the device with the same color facing the same side. Then the chesses will roll down to the exit. At the bottom of the device, the first chess will roll out along an inclined surface with the opponent’s color facing upward. While the chess is not taken away, it will cause the other chesses to be stuck so they will not roll out until the bottom chess is removed. However, this is only a
brief design of the prototype which has not gone through any tests. Therefore, it may not work as imagined.

![Figure 14: Automatic chess roll out device prototype](image)

3.3.3 Improved Detection Method

For the detection, instead of searching for specific colors and shapes, methods like YOLO can be implemented. It is a method that uses different example images to train the detection. Then the objects with high confidence are detected. This kind of detection method is suitable for a specific aspect like this project where only the chessboard and chesses are required to be detected.

3.3.4 Robotic Arm

Finally, the robotic arm can be improved by replacing a new gear if the problem is just that the gear stripped. However, at the time this problem was found, it was almost the deadline of the project. Therefore, there is not enough time to verify if the problem is this or not.
4. Conclusion

The project aims to create a mobile app to control the robotic arm to play an Othello chess game with a human player. The mobile app is coded using Android Studio with Java to control the robotic arm. The robotic arm can move along 3 axes and pick up and put down object. The Arduino board receives commands from the app and controls the robotic arm.

However, there are several difficulties encountered in the progress, including the flipping action of the robotic arm and object detection. Therefore, there are still a lot of improvements can be made.
References


Figures

Figure 4: Hough line transform¶. Hough Line Transform - OpenCV 3.0.0-dev documentation. (n.d.). Retrieved April 18, 2022, from https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

Figure 5: Hough Circle transform¶. Hough Circle Transform - OpenCV 2.4.13.7 documentation. (n.d.). Retrieved April 18, 2022, from
https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough-circle/hough_circle.html

Figure 6: Canny edge detection. OpenCV. (n.d.). Retrieved April 18, 2022, from https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

Figure 7: Bourke, P. (n.d.). Points, lines, and planes. Point, Line, Plane. Retrieved April 18, 2022, from http://paulbourke.net/geometry/pointlineplane/

Other figures: Self prepared
Attachment A

Installation

Install OpenCV Manager on the smartphone

Download the source code from https://github.com/Chungpy/fyp21069

Download OpenCV 3.4.14

Open Android Studio and import the OpenCV library as module

Set the OpenCV library in the dependency list to utilize the functions provided by OpenCV

Then connect the smartphone to the computer and run the app

The Arduino code is not prepared in this project. It is previous work by others. The Arduino code can be found at https://github.com/hkuсs-makerlab/robotArm
Attachment B

1 Controls

The buttons functions are as follows:

“Open” button: the app will establish the socket connection with the Arduino board.

“Close” button: the app will close the socket connection with the Arduino board.

“Steppers” switch button: the app will turn on or off the stepping motors.

“Fan” switch button: the app will turn on or off the fan.

“Gripper Open” / “Gripper Close” buttons: the app will turn on or off the vacuum to draw out the air inside the tube to pick or place objects.

The number picker under “Gripper Open” / “Gripper Close” buttons: the app will set the value of the vacuum.

Positions buttons (“Home” / “Transition Flipper” / “Transition Chessboard”): the app will control the robotic arm to move to the original position when powered on, the transition position above the flipper and the transition position above the chessboard respectively.

Movements buttons (“X+” / “+” / “X-” / “-” / “Y+” / “+” / “Y-” / “-” / “Z+” / “+” / “Z-” / “-”): the app will control the robotic arm to move in the corresponding direction with short or long distance. For X-axis, “X+” moves the robotic arm to its right side for a long distance, “X-” moves the robotic arm to its left side for a long distance. “+” moves the robotic arm to its right side for a short distance, “-” moves the robotic arm to its left side for a short distance. For Y-axis, “+”
means forward and “-” means backward. For Z-axis, “+” means upward and “-”
means downward.

Calibration buttons:

“Calibration i” button: the app will record the current position of the robotic
arm and link it to the grid index i. For example, “Calibration 0” means recording
the top right grid according to the view of the robotic arm. From i=0 to i=7,
they are the grid from top right grid to top left grid. From i=7 to i=14, they are
the grid from top left grid to bottom left grid. In other words, from i=0 to i=14,
the positions recorded forms a rotated “L” shape containing the upper side
grids and left side grids.

“Flipper In” / “Flipper Out” buttons: the app will record the positions of the
entrance and exit of the flipper which are the positions that the robotic arm
place down the not yet flipped chess and pick up the flipped chess.

“New Chess” button: the app will record the position that the robotic arm picks
up a new chess to place on the chessboard in the robotic arm’s turn.

“Automatic” switch button: the app will turn on the automatic mode if
switched on and control the robotic arm automatically.

“AI First” switch button: the app will switch the first turn player to the robotic
arm allowing the robotic arm to make a move first if switched on. This switch
button only works before switching on the “Automatic” switch button.
2 Instructions / Steps

1. Start the app and turn on the robotic arm

2. Turn on Bluetooth and press “Open” button to connect to the Arduino board of the robotic arm

3. Hold the robotic arm with the front arm in horizontal position and the back arm in vertical position as shown in figure 15

4. Press the “Steppers” switch button to turn on the stepping motors of the robotic arm. Then the robotic arm should hold still at this position

5. Set up the smartphone stand, chessboard and chess

6. Check if the chesses are detected and are all the corners detected and no light reflection is affecting the detection

7. Press the movement buttons to move the end point of the robotic arm to the top right grid from the view of the robotic arm

8. Press the “Calibration i” button to record the grid position

9. Repeat step 7 and 8 until grids from top right to top left and from top left to bottom left are recorded (the order must follow from top right to top left to bottom left with total of 15 positions to record)

10. Move the end point of the robotic arm to the entrance of the flipper

11. Press “Flipper In” button to record the position

12. Move the end point of the robotic arm to the exit of the flipper

13. Press “Flipper Out” button to record the position

14. Move the end point of the robotic arm to the position of picking a new chess
15. Press “New Chess” button to record the position

16. Press “Transition Flipper” to check if the robotic arm moves above the exit of the flipper

17. Press “Transition Chessboard” to check if the robotic arm moves to the center of the chessboard

18. Press “Transition Flipper” to move the robotic arm to the standby position

19. Turn on the “AI First” switch button if the player wants to play second

20. Turn on the “Automatic” switch button to start the automatic mode

21. The player can now play with the robotic arm

22. Every time the robotic arm places a new chess, the player needs to refill the chess at the “new chess” position where the player recorded it

Figure 15: Starting position of the robotic arm
Figure 16: App user interface