Gamer Robot: Applying Pikaliya Chess Algorithm on a TM5-900 Robot Arm with an RGB camera

Kuan-Yu Su
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
r09921002@ntu.edu.tw

Yuan-Kai Chang
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
r09921004@ntu.edu.tw

Chih-Yian Chuang
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
r09921006@ntu.edu.tw

***Abstract*—In this paper, we propose a Gamer Robot that can play Pikaliya chess with humans. The rule of Pikaliya chess is a combination of Tic-Tac-Toe and Gomoku. Generally, two players are required. However, with Gamer Robot, even a single person can play. The robot consists of three main parts: a TM5-900 robot arm, an RGB camera, and a host computer. With these elements, we can build the Gamer Robot. The last thing we need is a robust Pikaliya chess algorithm to be applied to the robot. This algorithm is programmed in Python language and communicates with ROS. The details will be presented in the following article. The first two sections of the paper cover the background knowledge of Pikaliya Chess and surveys of several related works. Technical details like Camera calibration, frame transformation, object detection, and chess algorithms are in Section three. Then, we exert experiments to demonstrate the overall robustness and stability of the Gamer Robot (Section four and five). Finally, the conclusion and future works will be discussed in section six.**

Keywords—TM5-900 robot arm, gamer robot, Pikaliya chess, game theory, Human-robot interaction

I. INTRODUCTION

In the research field of human-robot cooperation, physical board games are a rich problem domain, because such games have an intermediate and easily adjustable degree of structure. Playing board games involves the perception of the board and chess pieces, reasoning about the game and game state, and manipulation of the physical chess pieces while coordinating with the human opponent. Progress on physical board game playing systems paves the way for more general human-robot cooperation systems that assume less structure. For example, this line of work could lead eventually to a manipulator capable of helping a chemist as a lab assistant that cooperatively performs manipulation tasks in an unstructured laboratory glovebox environment [1], [2].

This paper introduces *Gamer Robot*, a robot manipulator system that is designed to autonomously play a board game called *Pikaliya chess* against human opponents. Pikakiya chess was originally popular in some Indian tribes in the state of New Herzegovina. People used a cloth to make chessboards and stones of different colors. In the 16th century, the Spaniards brought Pikaliya to Europe, named after the Spanish word Pikaliya, which means "stone chess". It is like a combination of Tic-Tac-Toe and Gomoku, and with some modification. The chessboard is a three by three, nine-square grid, with cross lines, see Fig. 1. The game is played by two players, and each has three chess pieces. Players alternate turns to place their three chess pieces at any vacant squares on the board, but the center one cannot be placed in the first round. When all of the chess pieces are placed, players then take turns moving their chess pieces to any adjacent vacant squares. However, players cannot move the same chess piece that was moved in the previous round.

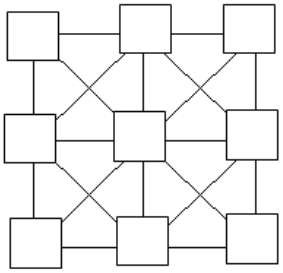

Fig. 1. Chessboard for Pikaliya chess

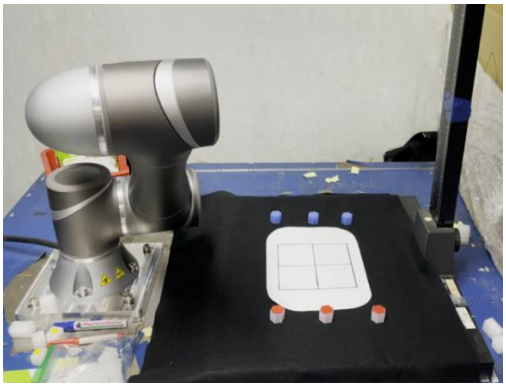

Fig. 2. Gamer Robot and Pikaliya chess setup

The objective is to attain, through placement or movement, three chess pieces in a straight unbroken chain. There are eight ways to connect the chess pieces, including vertical, horizontal, and diagonal lines.

In this paper, we focus on the game of chess, see Fig. 2. A large number of chess-playing automata that have been imagined or constructed in the last three centuries suggests that robot chess could be interesting as an entertainment application. Instead, we further view robot chess as a testbed problem of adjustable difficulty that can advance research in perception and manipulation in a noisy, less constrained real-world environment. For this reason, we choose Pikaliya chess as a task. Because different from other popular board games like Gomoku, Go, and Chess, which have all been studied extensively, Pikaliya chess is relatively uncommon and hasn't been implemented in any robotic system. Also, as mentioned, Pikakiya chess combines different board game properties. By choosing it as our task and building all the algorithms and manipulator system from scratch. We hope to add more variety and maybe some inspiration in such problem domain.

II. RELATED WORK

The Mechanical Turk, first exhibited in 1770, is perhaps the first purported chess playing automaton. It was in fact not autonomous at all, as it relied on a concealed human

chess master for perception, game logic, and control of the Turk's manipulation hardware, which consisted of a mechanical arm and hand, as well as a "voice box" that could say a single word ("Echec"). The Turk used magnetically instrumented chess pieces that enabled the operator to sense the game state via the motion of corresponding magnets in the operator's compartment. Thus the mechanical Turk was actually a chess teleoperation system, not a chess automaton [3].

With chess often being used as the research object, efforts are being made computers to have the same game capabilities. Then it will be able to think, judge and make rational decisions. For many years, researchers in the area of artificial intelligence have been working on algorithms to play chess automatically. Alan Turing and David Champernowne were the first to develop a program capable of playing a full chess game [4], known as "Turing's paper machine" [5], [6]. Because at the time there were no computers capable of executing the instructions, it was Turing himself who performed the processing tasks using paper and pencil. It was not until 1996 that a fully functional computer program, IBM's Deep Blue, was able to defeat the world champion Gary Kasparov.

Nearly all autonomous board game playing systems use instrumented or special purpose game boards and pieces to simplify perception and manipulation. Thus, they do not handle the complexities introduced by using arbitrary pieces, boards, and environments. For example, chess playing robot arms were shown in 2010 at the Maker Faire, and at several press events in Russia [7]. They used instrumented chess sets such as the Digital Game Technology Sensory Chess Board to eliminate the perception problem. A commercial product called the Novag 2 Chess Computer includes a robot arm that can play against a person. However, it uses a special instrumented board for perception and special chess pieces that are co-designed with the manipulator.

Compared to above related works, our Gamer Robot system can play with arbitrary chess sets on a variety of boards, requiring no instrumentation or modeling of pieces. Gamer Robot monitors the board state and detects what kind of move an opponent has made. Furthermore, Gambit's perceptual system tracks the board pos. The board is not fixed relative to the robot and is calibrated as game start.

III. METHODOLOGY

A. Camera Calibration

In the Gamer Robot system, we use a regular RGB camera as the sensors to gather information about the environment. In order to acquire undistorted images from the camera for physical board game environment perception, we need to first pre-process the raw image, see Fig. 3. This step is called camera calibration. It is the process of estimating the parameters of a pinhole camera model, such as focal length and principal point. This process is required for cameras before doing image processing.

To achieve this goal, we use the camera calibration package provided in ROS as the initial step of building the Gamer Robot system. This package allows easy calibration of monocular or stereo cameras using a checkerboard to calibrate the target. To start the calibration, we will first load the image topics that will be calibrated. Then the calibration

Fig. 3. Camera calibration, before (left) and after (right).

window will be opened up, which will highlight the checkerboard. In order to get a good calibration result, we move the checkerboard around in the camera frame. As the checkerboard being moved around, the program will start doing the calculation. After the calibration is complete we can get the calibrated image.

B. Frame Transformation

To combine the robot manipulator and camera into one system in Gamer Robot, we need to be able to transform the coordinate of a given point in the camera frame into the position of the robot arm's end effector. This process is called frame transformation, its goal is to derive a three by three transformation matrix T defined as follow.

Let $c = [c_x, c_y, 1]^T$ be a 3-dimension vector in which c_x and c_y represent x and y coordinate of a point in the camera frame respectively. And $r = [r_x, r_y, 1]^T$ be a 3-dimension vector in which r_x and r_y represent the x and y coordinate of the robot arm's end-effector corresponding to the same point in the camera frame. The relation between c and r is then defined by transformation matrix T, such that $r = A \cdot c$. With this equation, as long as we have the correct transformation matrix, the robot arm's end-effector can reach any given point in the camera frame by sending the transformed coordinate as a command.

At the beginning of building the system, we need to calculate the correct transformation matrix. First, we pick four corners of the board and acquire the vectors c and r of each. Then, with four $r_i = A \cdot c_i$ relation, we can concatenate these equations into $R = A \cdot C$, in which R and C are 3 by 4 matrices. To get the best approximation of the transformation matrix A, we can apply the least squares method, that is $A = (R \cdot C^T) \cdot (C \cdot C^T)^{-1}$.

C. Object Detection

Perception of the chess pieces is the first fundamental function of the Gamer Robot. So that the robot can update and reason about the game state. With the help of image processing, a robot can analyze the image of the immediate environment imported from the camera and use the result to determine the appropriate action to take. We use the OpenCV library as the image processing tools for detecting the current location of every chess piece. The details are as follows.

In our physical board game settings, one player uses the red chess pieces and the other uses the blue ones. So the task of image processing is to detect the center position of red/blue pieces. First, we convert the raw image from BGR to HSV color space. Then we use the self-defined color mask, red and blue in this case, to perform thresholding on the image. After this, we can get an image with only the desired color part, that is, the chess pieces. Finally, using

the built-in function in the OpenCV library, the contours and center location can be acquired.

D. Robot Arms Control

As mention before, we use OpenCV to get the frame transformation between the RGB-camera and TM5-900 Robot Arm. At the beginning of the game, we take a picture and find the board and chess pieces coordinate with RGB-camera. Then, use a transformation matrix to let those coordinates transfer to the robot frame.

We use three by three chessboard as our hardware, so we need to get nine corners on the board. goodFeaturesToTrack() is a function of OpenCV[8]. It finds corners in the image by the Shi-Tomasi method. This function will accept the corner if

$$\min(\lambda_1, \lambda_2) > \lambda \qquad (1)$$

where λ_1 and λ_2 are eigenvalues of the window, and λ is a predefined threshold. The window size is defined by us, and the function will choose one corner in each window. It can prevent us from choosing the same corner in our chessboard. All corners below the quality level are rejected. Then it can sort the remaining corners based on quality and take N strongest corners in the image. N is the number of corners we want to find. Because we use three by three chessboard, we defined N which is equal to nine.

We get nine corner coordinates of our chessboard by OpenCV. Then we sort those coordinates and give a number for each corner as Fig. 4. The detail process is as folllows: (1) First, we sort x coordinates. The three smallest x coordinates are assigned to the first row of the chessboard. The three largest ones are allocated to the third row of the chessboard. (2) In each row, we sort y coordinates to classify this corner into a specific column. (3) Then, we give a number to each corner. Each number will correspond with a pair of a coordinate of this corner. The advantage is that we can have some tolerance for our chessboard orientation.

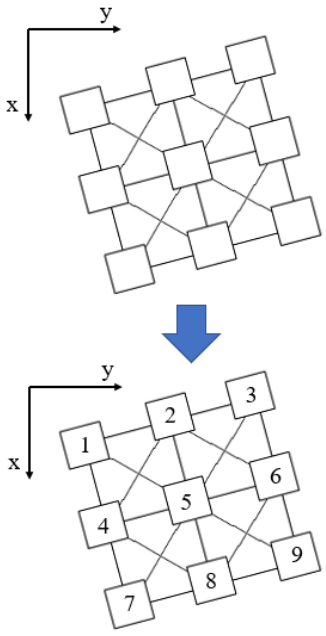

Fig. 4. Sort coordinates and give a number to each corner.

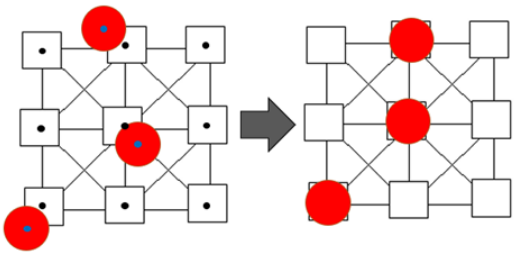

Fig. 5. Detect player chess pieces. Black dots are the coordinates of the corners. Blue dots are the center coordinates of the chess pieces.

Also, we can use these numbers to control the robot. Let the robot know the target corner where the chess piece should be placed.

For robot arm control. We use two different aspects in each part. In the first part, we need to place the chess pieces which are not on the chessboard. The robot can arbitrarily choose a piece from those and place it on a specific space. However, in the second part, we need to move a specific piece to an adjacent vacant space. It is hard for the robot to tell the difference between pieces with the same color because the position of their pieces would keep on changing. We use the coordinates of the chessboard corner which were recorded at the beginning of the game. The robot will choose a corner it wants to move and find the chess piece which is closest to the corner. Finally, the robot arm grabs the piece and places it on the target corner.

E. Detect Player Chess Pieces

Our objective is to let robot arm play board games with people. In order to accomplish this objective, the robot needs to detect the player chess pieces in each round. With an RGB-camera and OpenCV, the robot can get the player chess pieces and chessboard corners coordinates. We use the Euclidean distance D_i between a piece and a corner.

$$D_i = (P_x - C_{ix})^2 + (P_y - C_{iy})^2 \qquad (2)$$

Where D_i is the Euclidean distance, P_x and C_{ix} are x coordinates of the player chess piece and the i[th] corner. Similarly, P_y and C_{iy} represent y coordinates. And, we define a threshold δ. If D_i is smaller than δ, then we assume that this piece is on this corner, see Fig. 5. The advantage is that we don't need to limit the player to place the piece on the accurate position. With this function, the process of the game can run smoothly.

$$P \in C_i \quad if\ D_i \leq \delta \qquad (3)$$

F. Algorithm

In this section, the algorithms we developed are presented. The algorithm will let the robot arm know how to play Pikaliya chess. Because Pikaliya chess is combined with two parts, so we developed different algorithms corresponding to each part. The first part is *Placing Algorithm*, it will let the robot grab the chess pieces outside of the board and place them on the desired corner. The second part is *Moving Algorithm*, the robot will grab a piece that is on the board and move it to an adjacent corner.

Algorithm 1 Placing algorithm

```
1: let player chess piece value be 1
   let robot chess piece value be -1
   let empty space value be 0
2: for chess piece ← 1 to 3 do
3:     while board center != 0 and the others == 0 do
4:         the first player violates the rule
5:     if the sum in any line == -2 then
6:         place the last piece in empty space in this line
7:     else if the sum in any line == 2 then
8:         block the player
9:     else if the sum in any line == -1 then
10:        if no empty space in this line then
11:            continue
12:        else
13:            place the chess piece nearby their pieces
14:    else
15:        do random choice
16:    chess piece ++
```

Moreover, we also develop an algorithm to detect the victory condition. It will check the situation and determine who is the winner after every move.

(1) Placing Algorithm

At the beginning of the Pikaliya chess, all the pieces are outside of the chessboard. Therefore, the robot arm can arbitrarily grab one of them. Algorithm 1 shows how the robot plays Pikaliya chess in the first part, including how to detect the violation we defined.

Line 3 prevent the player from violating the rule which the first player can't place the piece in the middle point in first round. Our method is to detect the value of all corners. If the player chess piece is on this corner, then the value of this corner will be 1, and the robot chess piece value is -1. If there is no piece on this corner, then its value will be 0. Hence, the robot can use these values to detect whether the player violates the rule or not. If violation happens, the robot will ask the player to place the piece on another corner.

The robot also can use these values to make some strategy decisions. Our chessboard is three by three, and it has three vertical lines, three horizontal lines, and two diagonal lines. We will calculate the sum of the value in each line. If the sum is equal to -2, it represents that the robot has placed two pieces in one line, and the robot will place the last piece on the empty space in this line, see Fig. 6(a). However, if the sum is equal to 2, it means that the player has placed two pieces in one line. The player will win the game in the next round, so the robot needs to block the player, see Fig. 6(b). If the player or the robot are not going to win, then the robot will place the chess piece nearby its other pieces to increase its chance of winning, see Fig. 6(c).

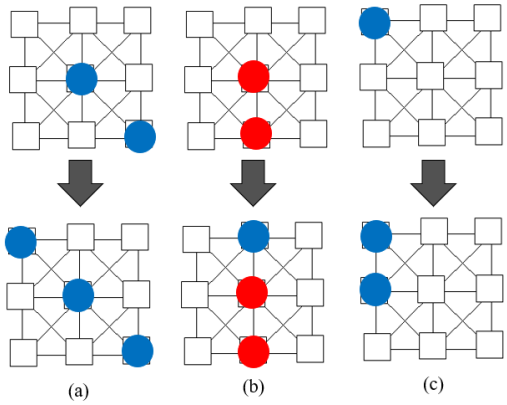

Fig. 6. Robot decision making. The red circles represent the player's chess pieces and the blue ones represent the robot's.

Algorithm 2 Moving algorithm

```
1: procedure
2: let player chess piece value be 1
   let robot chess piece value be -1
   let empty space value be 0
   P ← the chess piece moved in previos round
   T ← the chess piece moved in this round
   S ← the space choosed in this round
3: the direction D = {UP, DOWN, LEFT, RIGHT
       UP_LEFT, UR_RIGHT, DOWN_LEFT, DOWN_RIGHT} 1:
4: Find the robot chess in the board
5: Try to move the chess T != P in D in every direction
6:     if robot is going to win == True then
7:         robot wins
8:     else if player is going to win == True then
9:         if chess T is nearby the empty space in this line then
10:            block the player
11:        else
12:            robot gives up
13:    else
14:        do {
15:        random choice
16:        } while(player is going to win == True)
17: From step 6 to step 16, robot would choose a chess T and space S
18: Move T to S in each round and update P
19: Repeat step 4 to step 18
```

(2) Moving Algorithm

Each player has three chess pieces. When all of the chess pieces are placed. The robot will execute Algorithm 2 automatically. This algorithm will let the robot know how to move their chess pieces and how to detect whether the player moves the same chess piece that was moved in the previous round.

To let the robot knows which chess piece it can not move. The robot will record the vacant square where it placed its chess piece in the previous round. Therefore, the robot can't move the chess piece which is closest to this square in this round. To prevent the player from moving the same pieces, robot will detect the positions of all player pieces. Then, use Logical AND operation to get two stationary chess pieces in this round. In the next round, the player has to move one of the two pieces, or the robot will ask the players to restore the move and wait for them to move chess pieces again.

To prevent the robot from making an unwise decision, we developed a function to let the robot predict whether the player would win the game in the next round. If there are two player's chess pieces in the same line and the last chess piece which the player can move is nearby the last vacant square of this line, the function will return true, and the robot will cancel this

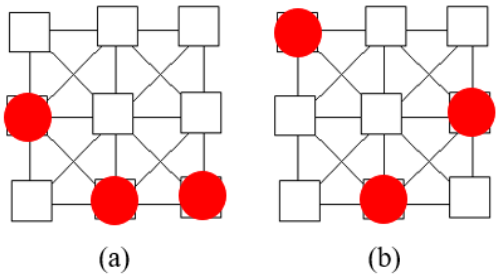

Fig. 7. Prediction function (The red circles mean the player's pieces). (a) return true (b) return false

Algorithm 3 Check Victory Condition

```
let player chess piece value be 1
let robot chess piece value be -1
let empty space value be 0
if the sum in any line == 3 then
    player wins
if the sum in any line == -3 then
    robot wins
```

decision, see Fig. 7.

Line 6 to line 16 show the procedure of making the decision. First, the robot will check its chess pieces, and judge whether it is able to win in this round. If the judgment result is true, then the robot will move the piece and win this game. Second, if the judgment result is false, then the robot will check the player's pieces, and judge whether it needs to block the player. Finally, if the robot can't block the player, then the robot will give up the game and surrender.

(3) Check Victory Condition

Algorithm 3 check the victory condition. Each player has different color pieces. Each color has a different value. There are eight ways to connect the chess pieces, including vertical, horizontal, and diagonal lines. If the sum of the value in one of them is equal to 3, it represents that the player places all pieces in an unbroken chain and vice versa. This algorithm will determine who is the winner.

IV. Experimental Setup

The robots consist of three main parts. First and foremost, the TM5-900 robot arm, its payload is 4 kilogram with a 900-millimeter reach, the typical speed is 1.4 (m/s), and 0.05-millimeter repeatability (repeatable accuracy), see Fig. 8(left). Second, the RGB camera, which acts as the visual sensor. We use a Logitech c310 webcam. The max resolution is 720/30 (p/fps) with 60 degrees diagonal field of view (dFoV) , see Fig. 8(right). Last but not least, a host computer with Ubuntu 16.04 operating system and ROS Melodic. For the architecture of the robot, see Fig. 9. The test site is in an indoor laboratory at the Department of Computer Science and Information Engineering (CSIE) building B1.

Fig. 8. TM5-900 robot arm (left) & Logitech c310 webcam (right)

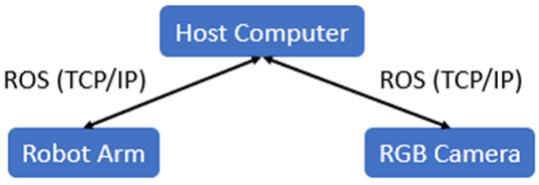

Fig. 9. The architecture of the Gamer Robot

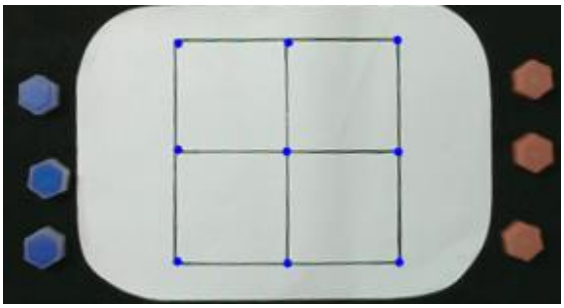

Fig. 10. Chessboard corner detection

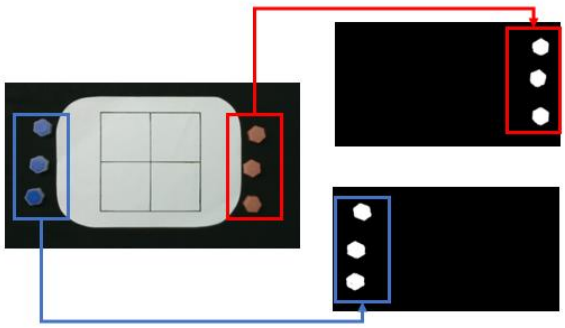

Fig. 11. Recognize different color (Red vs Blue).

V. Results and Discussion

A. OpenCV Detection Test

By applying OpenCV, the Gamer Robot can locate the intersections of the chessboard. Thus, it knows where to put its chess pieces. Furthermore, to understand the whole board situation. The robot also has to distinguish the color of each chess piece. With the RGB detection function provided by OpenCV, the robot now sees the whole picture as the human player does, see Fig. 10. and Fig. 11.

In most of the conditions, the robot works finely. However, illumination problems still occur sometimes since the environment is not strictly under control. To minimize the effect of the light, we picked the largest threshold values for RGB detection. Therefore, it enhances the robustness of the Gamer Robot.

B. Algorithm and Rule Violation Test

With the help of the OpenCV database and RGB camera, the robot can see the whole chessboard. The next step is controlling the robot arm to the designated location to take or put a chess piece. The main issue here is how to transform the coordinate from the camera to the robot arm since their original coordinates are different. The details are in section III calibration part. Then, to ensure the robustness of the algorithm. The robot was experimented with by actually playing with different people, see Fig. 12. Rule violation detection is also a feature that we want to implement on the robot. To be specific, players should not put chess pieces in the middle of the chessboard in the first round if they take the initiative, see Fig. 13. Also, consecutively moving the same chess piece is banned at all times, see Fig. 14. Thus, we tested the above detection functions as well. In practice, the algorithm is programmed in Python language and connects to the robot arm with TCP/IP protocols.

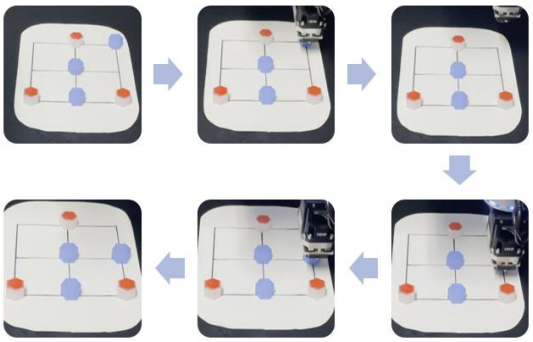

Fig. 12. The Gamer robot is playing with one of our team members.

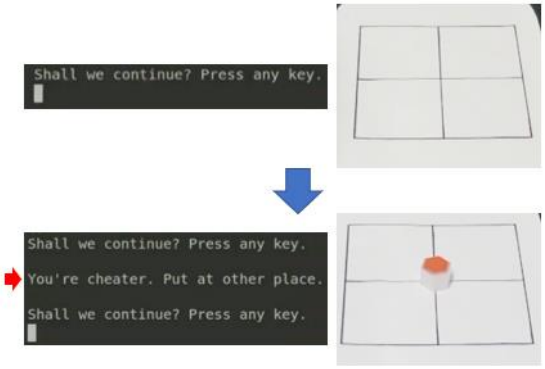

Fig. 13. Violate the rule that the player with initiative can't put in the middle in the first round. The left side are messages from the terminal, while the right side is actual chess piece.

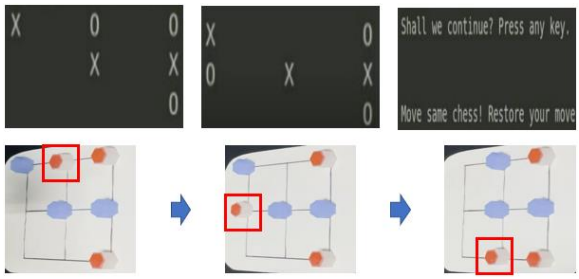

Fig. 14. Violate the rule that never consecutively moves the same chess piece.

Although, there's only one main chess algorithm implanted in the robot. It's compatible with most of the strategies exerting by human players. Statistically, the robot has a higher winning rate (above 50%) while facing a newcomer. When people get familiar with the rule and some critical tips, like which move is a must, the winning rate drops but still can remain around 30%. It is because people make wrong decisions sometimes. The result shows that this algorithm is stable enough to apply on the Gamer Robot.

VI. Conclusion and Future Work

In this paper, we present the idea of a robot that can play Pikaliya chess, which is the Gamer Robot. With ROS camera calibration toolkits and OpenCV library. The robot can obtain the correct camera coordinates for the chess pieces. Then, an accurate frame transformation matrix enables the robot to reach the target piece. Combine the above techniques with the Pikaliya chess algorithm, which separates into placing and moving sub-algorithm. We fulfill the prototype of the Gamer Robot. To further refine our algorithm, rule violation detection methods are being added. Finally, we successfully demonstrate the overall robustness and stability of the algorithm. The experiment results show that this robot has enough winning rates to compete with human players.

Moreover, the technique is a demonstration of human-robot interaction. The robot doesn't need to be constrained to board games. It can be a challenge to apply in the medical field or for other purposes that required high accuracy. For instance, a surgical robot is becoming a trend recently. In summary, the robot we present is a beginning. Many open problems and research issues are there, waiting for a solution.

References

[1] C. Matuszek et al., "Gambit: An autonomous chess-playing robotic system," 2011 IEEE International Conference on Robotics and Automation, Shanghai, pp. 4291-4297, 2011.

[2] Burger, B., Maffettone, P.M., Gusev, V.V. et al., "A mobile robotic chemist," Nature, vol. 583, pp. 237–241, 2020.

[3] T. Standage, "The Turk: The Life and Times of the Famous Eighteenth-Century Chess-playing Machine", Walker, 2002.

[4] Crandall, Jacob W., et al., "Cooperating with machines," Nature communications, vol. 9.1, pp. 1-12, 2018.

[5] Kasparov, Garry, and Frederic Friedel., "Reconstructing Turing's" paper machine"," J. Int. Comput. Games Assoc., vol. 40.2, pp. 105-112, 2018.

[6] van den Herik, H. Jaap, "Computer chess: From idea to DeepMind," ICGA Journal, vol 40.3, pp. 160-176, 2018.

[7] M. D. Anderson, S. Chernova, Z. Dodds, A. L. Thomaz, and D. S. Touretzky, "Report on the AAAI 2010 Robot Exhibits," AAAI Magazine, in press, 2010.

[8] Jianbo Shi and Carlo Tomasi, "Good Feature to Track," 1994 IEEE Conference on Computer Vision and Pattern Recognition Seattle, June 1994.

[9] https://www.logitech.com/en-us/products/webcams/c310-hd-webcam.960-000585.html?crid=34

[10] http://www.ia.omron.com/products/family/3739/specification.html

Distribution of Work

- Common work:

Topic discussion, environment setup, algorithm development, experiment, video filming, progress presentation, report writing, and demo are all finished collaboratively.

- Kuan-Yu Su (r09921002):

Responsible for *Abstract*, *Experiment Setup*, *Results and Discussion*, and *Conclusion and Future Work* in this report.

- Yuan-Kai Chang (r09921004):

Responsible for *Methodology Part. D Robot Arm Control*, *Part. E Detect Player Chess Pieces*, and *Part. F Algorithm* in this report.

- Chih-Yian Chuang (r09921006):

Responsible for *Introduction*, *Related Work*, *Methodology Part. A Camera Calibration*, *Part. B Frame Transformation*, and *Part. C Object Detection* in this report.

- Demo Video Link (Youtube):
https://youtu.be/FogqWetMIyk