C.H.E.S.S.
Chess Helper for Elderly with Speech-recognition System

1st Chandra, Benedictus Kent
CSIE - R11922178
National Taiwan University
Taipei, Taiwan
bkentchandra@gmail.com

2nd Liawi, Felix
CSIE - R11922177
National Taiwan University
Taipei, Taiwan
felixliawi@gmail.com

3rd Chun Hao, Yu
CSIE - R11922150
National Taiwan University
Taipei, Taiwan
howard89213@gmail.com

4th Jia He, Lin
CSIE - R11922048
National Taiwan University
Taipei, Taiwan
peterfocs3@gmail.com

***Abstract*—Technology has been advancing very rapidly in the recent years. Through the invention of internet, we are now more connected than ever before. Younger generations are more confident and avid into trying new things, however older generations struggle to keep up. With the recent COVID-19 outbreak, it became relatively hard to meet friends, family, or loved one physically and elderly who struggles to keep up with the technology got hit the hardest. In this paper, we propose a novel method to play chess using TM5-900 arm robot to give the sense of playing real chess and speech recognition system as the movement control. Through speech recognition system that is connected online, elderly can play chess with their friends, family, or loved one in the comfort and safety of their own home. The code and video for this project can be found here: https://drive.google.com/drive/folders/1KmVbS3516pTpfrC50ZK9wZHIkrwXrrWR?usp=share_link.**

***Index Terms*—transformation matrix, speech recognition, chess, cloud**

I. Introduction

The advancement of technologies using cloud in recent years are far from beyond along with the COVID-19 outbreak that makes the development even faster. Most people use remote methods to collaborate with their co-workers, playing physical games remotely with their friends, or having a conversation with their loved one. But the old generation is not familiar with this advanced technology. Hence, in this paper, we propose a remote Chess Helper for Elderly with a Speech-recognition System (C.H.E.S.S) using a TM5-900 arm robot to give the sense of playing real chess. The design of the C.H.E.S.S user experience is very easy which is to make sure that old generation people can play it easily.

II. Related Work

In this section, we will discuss the relevant research done on transformation matrix and Jaro-Winkler, a string comparator method used in the experiment.

A. Transformation Matrix

Ganapathy [2] explained that the relationship between the three dimensional coordinates of a point and the corresponding two-dimensional coordinates of its image, as seen by a camera, can be expressed in terms of a 3 by 4 matrix using the homogeneous coordinate system. This matrix is known more generally as the transformation matrix and can be determined experimentally by measuring the image coordinates of six or more known points in space. Such a transformation can also be derived analytically from knowledge of the camera position, orientation, focal length and scaling and translation parameters in the image plane.

B. Jaro-Winkler

Jaro–Winkler distance is a string metric measuring an edit distance between two sequences. This method is proposed by Winkler [5] which is a variant of the Jaro distance metric by Jaro [3]. The Jaro–Winkler distance uses a prefix scale p which gives more favourable ratings to strings that match from the beginning for a set prefix length l. The higher the Jaro–Winkler distance for two strings is, the less similar the strings are. The score is normalized such that 0 means an exact match and 1 means there is no similarity. The original paper actually defined the metric in terms of similarity, so the distance is defined as the inversion of that value ($distance = 1 - similarity$).

The Jaro similarity sim_j of two given strings s_1 and s_2 is

$$sim_j = \begin{cases} 0, & \text{if } m = 1 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right), & \text{otherwise} \end{cases}$$

where $|s_i|$ is the length of string s_1, m is the number of matching characters, and t as the number of transpositions. Jaro similarity score is 0 if the strings do not match at all, and 1 if they are an exact match. In the first step, each character of s_1 is compared with all its matching characters in s_2. Two characters from s_1 and s_2 respectively, are considered matching only if they are the same and not farther than $\left\lfloor \frac{\max(|s_1|,|s_2|}{2}) - 1 \right\rfloor$ characters apart.

Given two strings s_1 and s_2, their Jaro-Winkler similarity sim_w is:

$$sim_w = sim_j + lp(1 - sim_j)$$

where sim_j is the Jaro similarity for strings s_1 and s_2, l is the length of common prefix at the start of the string up to a maximum of 4 characters, and p is a constant scaling factor for how much the score is adjusted upwards for having common prefixes. Jaro-Winkler distance d_w is defined as $d_w = 1 - sim_w$.

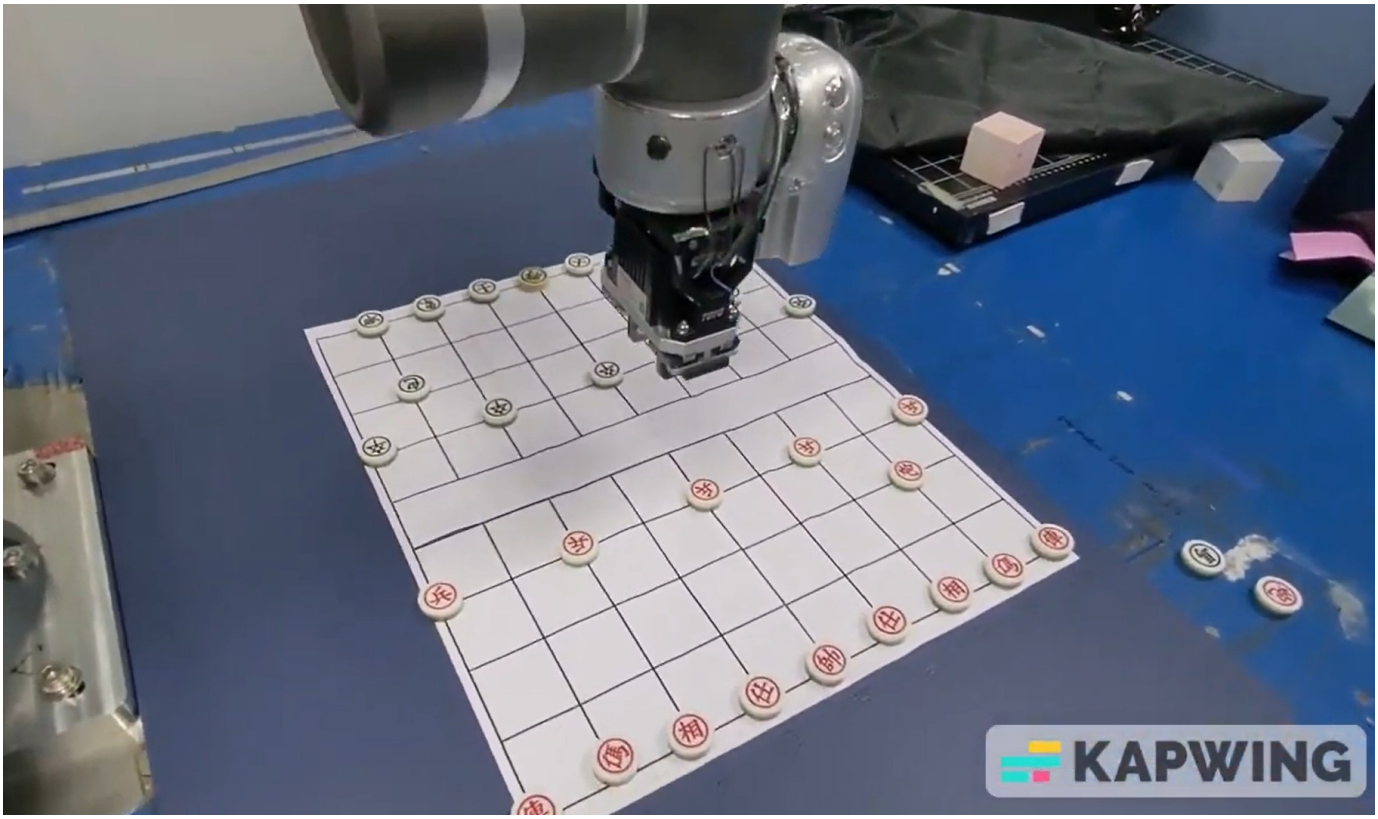

Fig. 1. Setup of the proposed method.

III. Methodology

A. Transformation Matrix

In order to get transformation matrix, we need to randomly assign several points in the 3D coordinates and take a picture of the space. In this research, we set Z height of the arm as 800 and used a cube as the points in space, thus we need to get the center point of the cubes. The center points need to be matched with x and y position of the cube in the image. Lastly, we only need to calculate the final transformation matrix.

$$T \cdot P_C = P_R$$

$$T = P_R \cdot P_C^-$$

T is the transformation matrix of robot arm, P_C is the position of points in camera perspective, and P_R is the position of points in 3D space. The details of P_C and P_R are listed below.

$$P_C = \begin{bmatrix} 632.6605 & 943.4092 & 428.2298 & 768.1837 \\ 667.6789 & 550.2138 & 441.6124 & 197.8780 \end{bmatrix}$$

$$P_R = \begin{bmatrix} 230 & 400 & 235 & 470 \\ 230 & 153 & 400 & 360 \end{bmatrix}$$

The final matrix we have is:

$$T = \begin{bmatrix} 4.036 \times 10^{-1} & -3.954 \times 10^{-1} & 2.376 \times 10^{2} \\ -3.970 \times 10^{-1} & -3.910 \times 10^{-1} & 7.425 \times 10^{2} \\ 0.000 & 1.301 \times 10^{-18} & 1.000 \end{bmatrix}$$

B. Localization

To have a better stability when grabbing the pieces, we design a sequence of localization process to avoid grabbing failure caused by calculation errors. In this part, we utilize lots of function in the OpenCV package to implement the localization function.

1) Board Localization: First we find the contour of the board using the $findContours$ function. Then, we use the $minAreaRect$ to obtain the minimum bounding rectangle of the board contour so we can get the four corner coordinate, width, height, and the rotation angle of the board. Finally, we can build the board coordinate according to those information.

2) Pieces Localization: After we done the board localization, we will take this as a reference. When we need to localize a piece, we first let the arm move to the position of that coordinate and a lower height. Then, we will perform the piece detection at that position so that the viewable range is much smaller and the accuracy can be higher.

C. Chinese Chess Algorithm

To make sure a chess game to proceed smoothly, and to avoid system crashes caused by illegal instructions from users, we decided to implement a chess game algorithm with Python to maintain the rules of the game. Besides, we also have a simple user interface of the chess game that allows

users to play online.
The input of the algorithm is a movement in the format: $source\_x, source\_y, action, target\_x, target\_y$, and the output will be the validity of this movement.

1) Pieces Object: A $pieces$ object has four attributes, including row, col, $color$, $board$.

row, col: to record the board coordinate of a piece.

$color$: to record whether a piece is red or black.

$board$: for convenience, each piece can access the board state through this attribute easily.

All types of pieces inherit from the $pieces$ object and have an is_valid function where we define the valid movement of the pieces. And we also define the $\_\_repr\_\_$ of each type of pieces for the visualization of our user interface.

2) Initialize Function: In this function, we create a $9 * 10$ array $board$ and also initialize the position and color of all pieces. Then, we maintain two array $alive\_pieces$ and $dead\_pieces$ so that we only need to check the alive pieces when validity check.

3) Showboard Function: This function is to visualize a simple user interface. The user can see the current game state on the board and also the dead pieces of both sides.

4) Is_check Function: This function is to detect a check both before the movement and after the movement. If a check detected before the movement means that the movement will cause the users themselves a check, then it should not be a valid movement.

5) Perform Function: In this function, we check the validity of the input movement, including whether the source and target pieces exist, whether the source pieces belong to the current user's side, and whether this movement will cause a check, etc. If all the validity check are passed, we will send a message to the arm to perform the movement physically.

D. Speech Recognition

In this paper, we used Google Speech-to-Text to recognize speech. Originally, we proposed to use simple default speech-to-text Python library in collaboration with Jaro-Winkler method. However, during our experiment, the result of the recognized speech is not satisfactory (see Experiment). Thus, we changed our method to use off-the-shelf and the state-of-the-art Google Speech-to-Text API. The API has the same problem as the one we experimented with, but using their method we can use utilize model adaptation to improve the accuracy of the transcribed text. We can give a set of words or phrases to the model and give a weighted value to the model, so the model can prioritize to use the given set of words first then transcribe normally if no matches are found in the set of words. The result is the speech recognition system is very accurate in transcribing our speech.

E. System Integration

After we have every single module's methodology and algorithm, we need to integrate them into a single entity which is C.H.E.S.S itself. The way to integrate each module is by creating a bridge between every single service. This bridge can be like REST API, Stream, or etc. Below are the bridge for connecting every service: Command Receiver Machine $\rightarrow$ Remote Cloud using REST API Remote Cloud $\rightarrow$Computer connected to the ARM using SSE (Server Send Event) The main reason to use a remote cloud to integrate the whole system is because the computer connected to the ARM is connected to wire cable which we cannot connect to it from the Command Receiver Machine (Fig. 2).

To control the arm, we use pub sub system design (Fig. 3). After the computer connected to arm received the command, the file listening to the server will write a text command into a text file, then, a subscriber file will detect whether there is an update in the text file or not. If yes, it will run the send_script.py. To let send_script.py dynamically read the latest command, we have added several lines of code to ask the send_script.py to first read the text file before doing anything, hence, we don't need to compile send_script.py as long as there are no code changes.

IV. Experiment

A. Coordinate for Transformation Matrix

In order to get the proper Z height of the arm, we try to set it to the value of 730, 750, 770, and 800. First, we calculate the coordinate of each point in the pictures of different Z heights, then we use these points to calculate the transformation matrix of each Z height with 4 points in the 3D coordinates. TABLE I shows the data we get in this experiment.

TABLE I
Coordinate for Transformation Matrix

Z Height	**Coordinate**			
	Arm Space		***Figure Space***	
	x	***y***	***x***	***y***
730	230	230	634.4438	682.0044
	400	153	973.7757	553.5259
	235	400	406.0696	435.0121
	470	360	783.2145	163.6040
750	230	230	634.7826	677.5237
	400	153	964.2362	552.6794
	235	400	413.1933	438.2503
	470	360	779.4118	174.8017
770	230	230	634.0497	672.6619
	400	153	955.1360	552.1570
	235	400	417.9629	438.9492
	470	360	774.8320	183.5947
800	230	230	632.6605	667.6789
	400	153	943.4092	550.2138
	235	400	428.2298	441.6124
	470	360	768.1837	197.8780

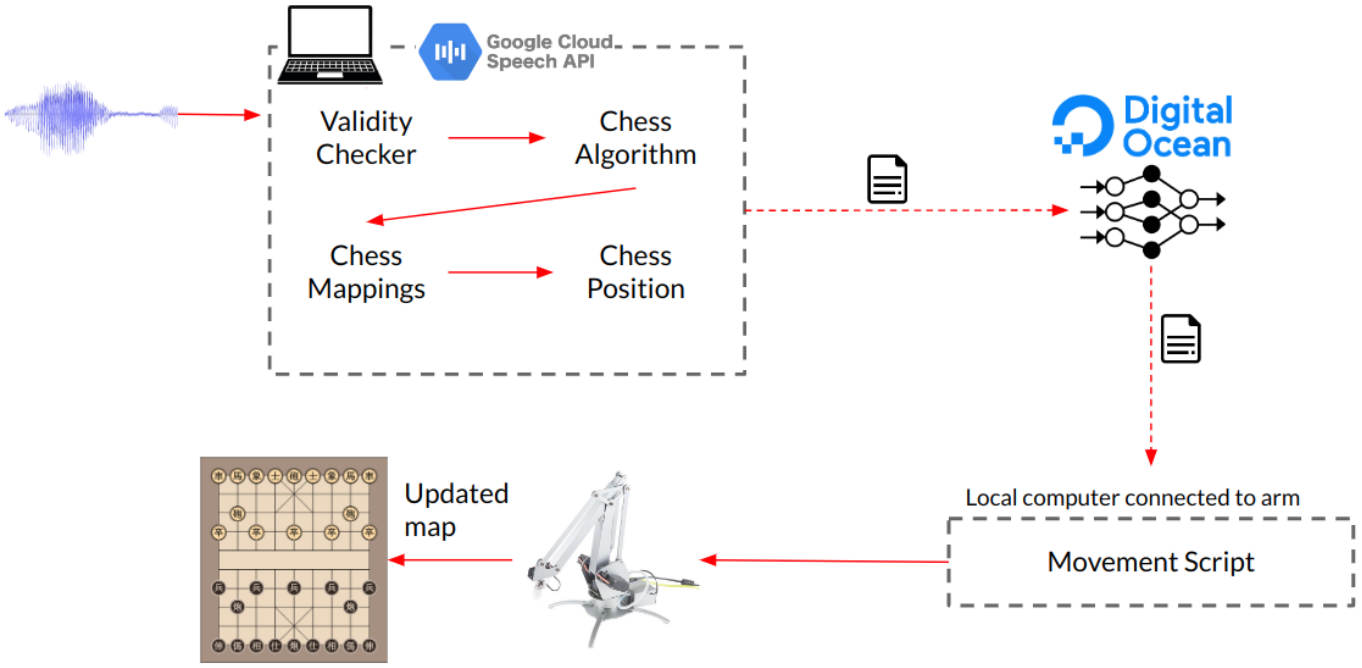

Fig. 2. System Integration.

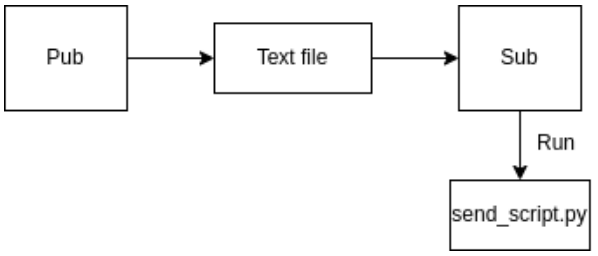

Fig. 3. Pub Sub Design.

The transformation matrix we calculate for Z height 730, 750, 770, and 800 is:

$$T_{730} = \begin{bmatrix} 3.654 \times 10^{-1} & -3.581 \times 10^{-1} & 2.424 \times 10^{2} \\ -3.611 \times 10^{-1} & -3.544 \times 10^{-1} & 7.008 \times 10^{2} \\ 4.337 \times 10^{-19} & -4.337 \times 10^{-19} & 1.000 \end{bmatrix}$$

$$T_{750} = \begin{bmatrix} 3.761 \times 10^{-1} & -3.692 \times 10^{-1} & 2.414 \times 10^{2} \\ -3.723 \times 10^{-1} & -3.657 \times 10^{-1} & 7.141 \times 10^{2} \\ 7.589 \times 10^{-19} & 1.301 \times 10^{-18} & 1.000 \end{bmatrix}$$

$$T_{770} = \begin{bmatrix} 3.871 \times 10^{-1} & -3.793 \times 10^{-1} & 2.397 \times 10^{2} \\ -3.807 \times 10^{-1} & -3.754 \times 10^{-1} & 7.239 \times 10^{2} \\ -2.168 \times 10^{-19} & -4.337 \times 10^{-19} & 1.000 \end{bmatrix}$$

$$T_{800} = \begin{bmatrix} 4.036 \times 10^{-1} & -3.954 \times 10^{-1} & 2.376 \times 10^{2} \\ -3.970 \times 10^{-1} & -3.910 \times 10^{-1} & 7.425 \times 10^{2} \\ 0.000 & 1.301 \times 10^{-18} & 1.000 \end{bmatrix}$$

B. *Speech Result*

To properly play chess using speech, we need to give a set of commands: piece and position. We achieved a promising result using just the vanilla speech-to-text in collaboration with Jaro-Winkler. The command can be, for example, cannon J8. Using only vanilla speech-to-text will usually mistranscribe the speech, for example cannon jake eight or cannon jade 8. The pattern of this library is that it tends to transcribe to words instead of numbers. We use the Jaro-Winkler to find the closest word to correct set of command and fix the command. However, this requires full understanding of the library tendencies, so we can anticipate what kind of errors the library will output.

Despite the promising result, we ran into a problem where the commands are not sufficient for us to know which piece they want to move and what action they want to do. There are 2 cannons or 5 soldiers in the field, using just the prior command it is challenging to know for sure which piece they want to move. We also need to know what action they want to do, either moving the piece or eating other piece. Therefore, we need to redo the speech recognition system.

Finally, the command that we need is <position> <move> <position>, for example J8 go A8. To eliminate inaccuracies, we changed alphabets into ICAO (International Civil Aviation Organization) radiotelephony [1]. Using this, the issue of mistaking alphabets as another is eliminated.

We ran into more problem with vanilla speech-to-text, that is, it starts to make unpredictable mistakes. As the command get longer, the more variations of mistakes it can make. This results in multiple different cases of mistakes with the same speech. Hence, it becomes more and more challenging to find

the patterns in the mistake. This was where we decided to change our method to Google Speech-to-Text.

We gave the model all the correct set of words and gave a weighted value to them. The result is that it gives high accuracy of transcription. We made sure that the commands needed are fulfilled and precise by adding another layer of security. This additional layer makes the command very flexible, meaning it does not have to follow the exact rule of the command. For example, the command can be <move> <position> <position> and it will still work.

C. Chess

After speech recognition has output the correct command in text, we use the output to check whether the move is valid. The validity checker is as explained in Methodology section. We will need to check:

- check which players' move,
- check whether player can move the piece,
- check whether the move is valid,
- check whether either player is checked,
- check whether a player is check-mated.

After the moves are analyzed, we put a feedback system to give players some input if they made an invalid move, or they are checked, and so on (Fig. 4).

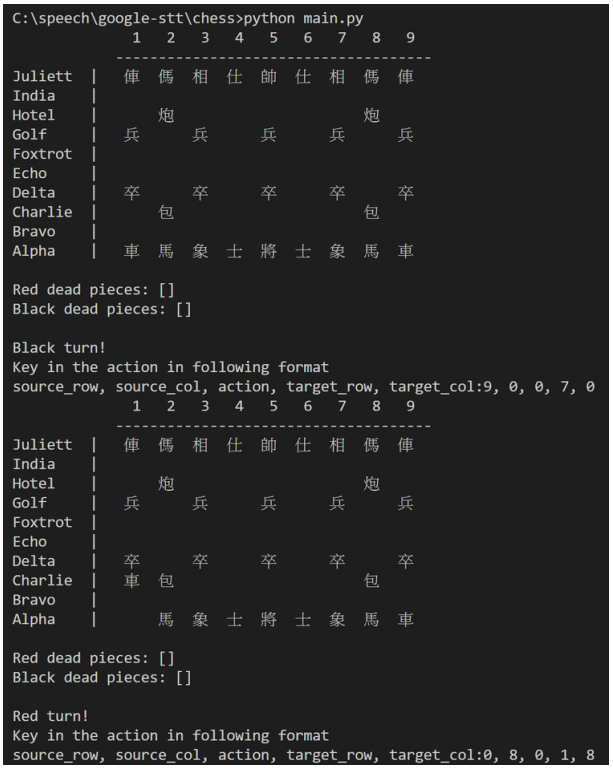

Fig. 4. Chess.

D. Speech Integrated with Chess

After going through speech recognition system and chess algorithm, we integrate them to do the experiment. Before every round, player need to say "command" to give command. If we give the system an **incorrect** command, it will tell us "I didn't get that, can you repeat it again?", and if we give it an **invalid** command, it will tell us "This move is invalid. Please try again!". Fig. 5 shows how it is working.

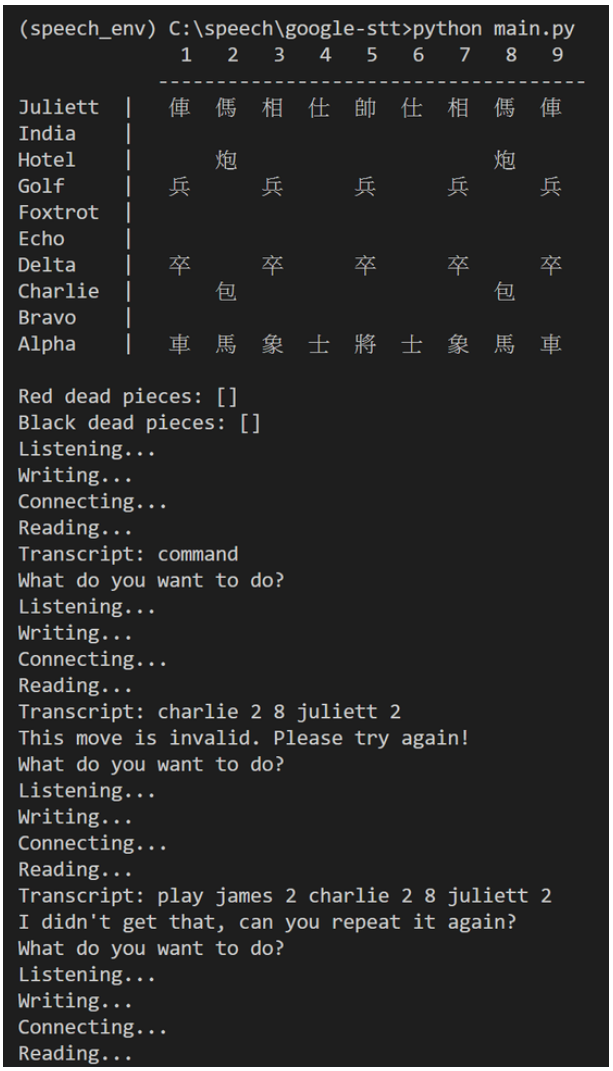

Fig. 5. Speech Integrated with Chess Algorithm.

E. Full System

We first need to receive the speech input from the user through a device by running the speech recognition system. Once the command are given, it will process the command and check whether the the move is valid. If the commands given invalid or incorrect, the system will give feedback to the player on the mistake. On the other hand, if the commands given are already correct, then it will let the player know that their command is correct and the robot arm is going to move the piece.

In the back-end, the correct command will communicate with the cloud server through an API and it will let local computer know about the move by writing the move in a text

file. The text file will be continuously processed by sub.py to check whether a new command has been given. Once the new command is inserted, the file sub.py will run the script to move the arm. There are 2 possible moves, go and eat. If the move is ”go”, the arm will grab the piece in the original position to the designated position. On the other hand, if the move is ”eat”, the arm will first grab the piece that is eaten and move it to the side. Next, it will finally grab the piece to the designated position. Once the robot arm is finished moving, it is now ready again to accept another command and

V. Conclusion

C.H.E.S.S. automatically extracts the oral commands from player, and executes the moves the player wants to make. Furthermore, we build a chess algorithm in it, therefore any invalid command will be found before the arm robot move the chess piece. Our main goal is to build a system for old generation playing Chinese chess with others, and C.H.E.S.S. can perfectly achieve this goal.

VI. Contribution

This section shows the contribution of the whole team in this project.

Name	Transformation Matrix	Chess Algorithm	Speech Recognition	Cloud	Integration
Chandra, Benedictus Kent	✓	-	✓	-	✓
Liawi, Felix	✓	-	✓	✓	✓
Chun Hao, Yu	✓	✓	-	-	✓
Jia He, Lin	✓	✓	-	-	✓

References

[1] Alphabet - Radiotelephony, www.icao.int/pages/alphabetradiotelephony.aspx.

[2] Ganapathy, S. “Decomposition of Transformation Matrices for Robot Vision.” Proceedings. 1984 IEEE International Conference on Robotics and Automation, 6 Jan. 2003, doi:10.1109/robot.1984.1087163.

[3] Jaro, Matthew A. “Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida.” Journal of the American Statistical Association, vol. 84, no. 406, 1989, pp. 414–420., doi:10.1080/01621459.1989.10478785.

[4] “Jaro–Winkler Distance.” Wikipedia, Wikimedia Foundation, 29 Nov. 2022, en.wikipedia.org/wiki/Jaro

[5] Winkler, William E. “String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage.” Proceedings of the Section on Survey Research Methods, 1990.