Intelligent Chessboard Using IOT

[1]Tejas Shaha, [2]Deepak Chaudhari, [3]Ashutosh Kshirsagar, [4]Aboli Doiphode, [5]Prof.M.R.Mahajan
[1]B.E Student, [2]B.E Student, [3]B.E Student, [4]B.E Student, [5]Professor
[1]Department of Information Technology
[1]Pune Vidyarthi Griha's College of Engineering and Technology, Pune, India

Abstract : A game of chess is both, fun and challenging. But lets face it, its a busy world. People stay miles away from each other and wanting to play with a partner might be inconvenient and infeasible. Automatic Chessboard enables you to play chess with the Intelligent chessboard anytime anywhere. The automated board is designed to reflect the Artificial Intelligence's move with precision. Intelligent chessboard consists of a Artificial Intelligence move-reflecting robotic arm which has 4 degrees of freedoms. Project is comprised of various modules such as; main controller, image processing, machine learning,serialization and motion engine of robotic arm. Image processing unit is triggered only whenever human player initiates a move. Meanwhile, images acquired in a specific time intervals are transmitted to the machine learning module for piece identification and classification. This process is followed by an Artificial Intelligence move computed by the Artificial Intelligence chess engine 'Sunfish'. Using serialization, robotic arm reflects this move on the board.

keywords **– Artificial Intelligence,Machine Learning,Robotic arm,Image Processing,Arduino,Sunfish.**

INTRODUCTION

As knowing, chess game is considered as the one of the excellent games which used to test the abilities and intelligence of human brains. It was developed in India and played for many years due to its attributes which can help in many fields such as putting military plans. Nowadays,chess game became as the one of the most important sports activities in schools. Furthermore, it enables the players from playing the game individually; however, sitting for a long time on the computer devices has several disadvantages such as damaging the sight, headache and etc. Intelligent chessboard which deals with chess pieces according to its attributes. It uses simple and low cost electrical circuit in designing.

I. EXISTING SYSTEM

In recent years, many papers and projects are published about computers and chess. Considering developments in computer vision, numerous papers are published within the field of recognition and detection of chess pieces. Despite that, a number of the papers are published with the aim to improve automation of the game.

Usually, camera-based systems for visual inspection require adequate and constant level of light source without shadows from surrounding objects. Therefore, overhead camera is employed in [1] with ambient lightning to detect player's moves. Human and chess piece shadows caused difficulties in move detection. Like a chess game, Janggi chess board detection under severe conditions in camera point of view is described in [2]. Using Hough transform, Harris corner detection and Canny edge detection, Janggi chess board and pieces are detected. Another camera-based approach is presented in [3] where authors created multimodal inter-face for making chess moves. Employing a stereo camera and difference in image entropy, hand gestures are detected and translated to chess moves. One approach in detection of chess position is supported based on player's hand movement (not piece's). Fully autonomous chess playing robot "MarineBlue" is presented in [4]. It combines elements of computer vision, chess engine and robot control. Authors in [5] implemented detection of chess pieces using reed sensors that are triggered by the magnetic field from permanent magnet placed on the underside of every piece.

PROPOSED SYSTEM

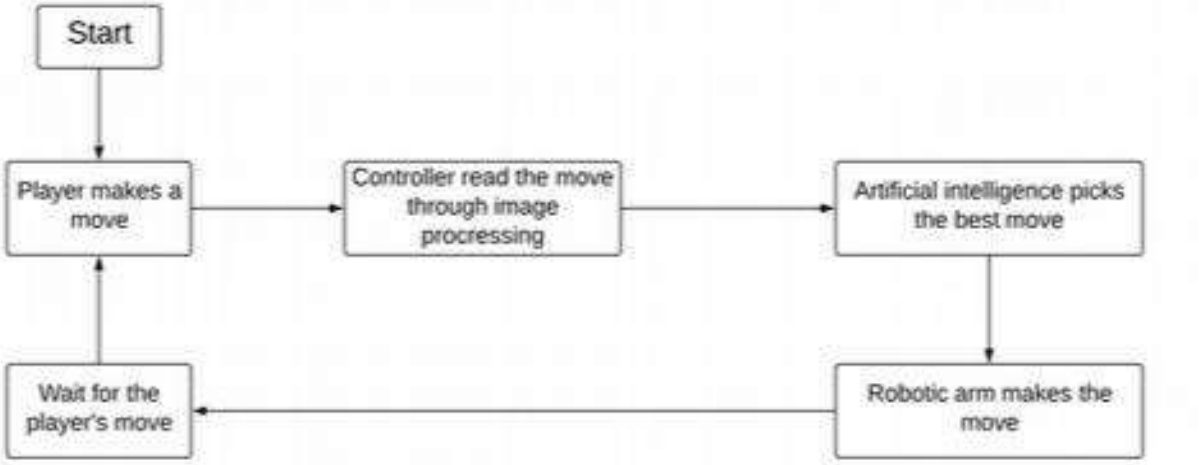

FIG.1 Architecture Diagram

In [6], there were some small problems. The metal disks they had chosen for the pieces were slightly bigger, and there were times when the magnet moved between the pieces and attracted either a different piece or multiple pieces at once. Also, the board they had used to place the pieces was not perfectly flat. Therefore, some parts of the board particularly the edges, were higher and the magnet sometimes couldn't attract and move these pieces. Another problem was that some chess pieces couldn't be placed directly on top of the reed switches because of imprecision. As a result the move could not be registered. So to reduce the complexity of underlying board, in our project we are using image processing. As shown in the above FIG [1], this system basically will detect the chess piece movements done by the player and then AI will make an appropriate move.The input to the system will be the old and new co-ordinates of chess pieces captured by overhead camera. These co-ordinates will be passed to the chess engine where the AI's next move will be calculated. Following this, the robotic arm reflects the move on chess board.

III. SOFTWARE

Chess Engine - Sunfish

Sunfish is a simple, but strong chess engine, written in Python, mostly for teaching purposes. Without tables and its simple interface, it takes up just 111 lines of code! Yet it plays at rating 1800-1900 at Lichess. It's simple open source chess engine under the GPL written by Thomas Dybdahl Ahle in Python for didactic purposes, inspired by Harm Geert Muller's Micro- Max. Sunfish supports the Chess Engine Communication protocol to play with a graphical interface like XBoard or PyChess.
The program uses Unicode glyphs to display the pieces within the terminal,making it look a little more chess- like than GNU chess. Moves are inputted by specifying the starting and ending co-ordinates,so the aggressive opening which moves the king's pawn to e4 would be inputted e2e4.Note that this can be slightly longer than the more common algebraic notation(in which the previous move would be written e4),but makes it much easier for computation. In this engine Lower case letters represent black pieces (p,r,n,b,k,q), and upper case letters represent white pieces (P,R,N,B,K,Q).Black-White-Empty matrix(BWE) is solely a listing of strings which represent each square on the board. This matrix is compared with an internally stored matrix within the Chess Engine.This suggests the Chess Engine can understand :

i)where the piece moved from

ii)where it moved to and construct a chess command from it. Moving pawn piece A2 to A4 at the beginning of the game would require command 'a2a4'.

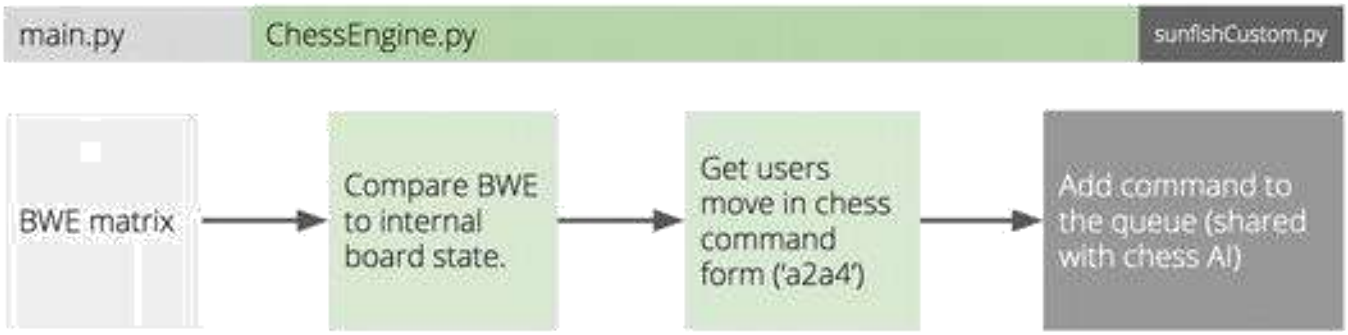

FIG.2 Chess Engine Flow

This move is trialed internally on the AI, and it responds with a number of options.
i) The move is invalid, in which case reporting of an illegal move back to the user needs to be carried out.
ii) The move is valid and has caused the user to win, in which case the user should be told.
Iii) The move is valid and the computer responds with move (which may be a checkmate).

The Chess Engine then splits this command into the start and end position of the move. It then converts these positions into indices,which it uses to search its internally stored board for the type of piece that is moving. In addition it checks if there is already a piece existing at the end position.

The Chess Engine first updates its internal board to remember the move the computer just made, then returns the following information to the caller (function) :

i) Firstly, if there is a piece to be killed, its location and type.
ii) Then, the start location of the piece moving, and its type.
iii) Finally, the end location of the piece moving.

IV. General Software Description

1. Image Processing

FIG.3 Webcam Image

Chess board detection under several conditions from camera point of view and image classification to each square is described in [7].The chess board in this image is skewed and rotated. The image taken may or may not be symmetrical but for the purpose of demonstration we will allow it. Also, we can't guarantee where in the image the board will be, or how skewed or rotated it will be. So, we'll need a way to find the board in the image and correct the perspective in order to identify the pieces. OpenCV implements many algorithms for image preprocessing as well as functions for identifying features like lines, corners, and contours. The first step in any computer vision task is to isolate the region of interest. There are a few ways we could approach this, but in this case, we have used contour detection.

First we will convert the image to grey scale in order to reduce the dimensionality. Next we will apply an adaptive gaussian filter to the image to make the board edges more pronounced. Contour detection is where we identify closed curves in the pixel data based on changes in pixel intensity. Basically, if we see white pixels next to black pixels then we calculate a closed path that separates them. In the plot below are both the filtered image and original image. On the left we can see that the features of the board are pretty well defined. On the right we have drawn green paths around all of the detected contours.

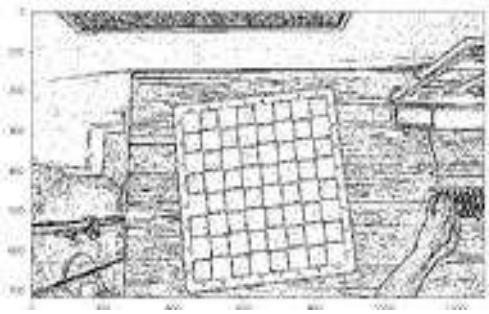

FIG.4.1 Grey Scale Image

FIG.4.2 Contours

OpenCV will also provide the hierarchy of identified contours, this saves us time as we don't have to calculate whether a contour is 'inside' of another contour in coordinate space. Now we can simply find the contour with the most children contours. This should be the inside edge of the board that surrounds the playing squares.

The green square in the image above is a path that is made up of the four corner points of the chess board. Next we can use these points to create a transform matrix to perspective correct and crop the image. OpenCV has functions to do this for us. We will need to create a source and destination matrix to apply this transform. The source is simply the four points from our board (we need to order them correctly) and the destination is a rectangle that is the size of the max distance of the ordered points.

Now that the board is perspective corrected and properly resized we can simply crop out any square we want. We will be doing this in a loop so let's create a function that takes the board image, the desired row and column, and returns the cropped image of the requested square.

Creating training images

To create the training we will loop through each square for each label. It starts at the square at (0,0) and when the space bar is pressed it saves the cropped square, moves to square (0,1) waits again for the space bar, and so on. We drew a green box on the current square and set the window title to display the current piece.
Then we moved each piece square by square at different rotations under different lighting conditions. This is where the training data can be improved.
We did this 4 times per piece at slightly different piece positions and looped through and empty board 4 times for empty squares. After this was finished, we programmatically rotated each image 90°, 180°, and 270° giving a total of 12 images per piece per square.

Creating model

Here we'll prepare the data for the model the same way we did above. This time we do it for the whole data set instead of just one sample.
We'll also break the data into XX and yy values. The XX data is the input (the image pixels of the selected square), and the yy data is what we will teach the model to predict (the index that corresponds to the correct label).
Next we will create categorical arrays. This means that for each XX we will have an array yy that is the size of our labels. We have 13 classes with corresponding labels:
['K', 'Q', 'R', 'B', 'N', 'P', '_', 'k', 'q', 'r', 'b', 'n', 'p']
so we need a yy for each XX that looks like this:
[0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]
This yy is for the label 'Q' because the 1 in yy in in the same position as the 'Q' in the label_map .
This what we're training our model to do. We'll give it an array of normalized pixels and it will give us back a value that looks like yy. For example, if the model returned:
[0.01, 0.2, 0.01, 0.9, 0.1, 0.02, 0., 0., 0.01, 0.1, 0., 0.02, 0.]
Then the model is saying the probability is .9 that the image is a white bishop ('B'). It's also saying that there is a 20% chance it's a white queen. We'll address this by choosing the index with the highest probability.
Finally, we will randomly select and set aside 20% of both the XX and yy values for validating the model.
Next we simply setup the data to have the correct shape and type. The final step it to normalize the pixels which are currently from 0 (black) to 255 (white) or some value in between (grey). We want these to be in the range of 0 to 1 to make the all of the values of the same scale.

The activation function used in this model are :

ReLU - Rectified Linear Units . All activation function have pros and cons. This is a good all around choice with many attractive mathematical properties.
Softmax - Softmax is a normalization function that is responsible for mapping the output of the model to

probabilities. Training the model

Training the model is a simple as calling model.fit. We'll configure this to use a batch size of 512, this is how many samples we want to train per epoch. Because were training this model in a notebook the model stays in memory and we can run fit multiple times to incrementally train.

Making Predictions (classifying chess pieces)
Later we can load the model from disk, loop through the board, get the playing square crop, and the model will predict the piece that is on that square.
Now we'll grab an image off of the disk and see if it correctly classifies the piece.

2. Arduino using Inverse Kinematics

In robotics and computer animation, inverse kinematics is the mathematical process of calculating the variable joint parameters needed to place the end of a kinematic chain, such as a robot manipulator or animation character's skeleton, in a given position and orientation relative to the start of the chain. In robotics, inverse kinematics makes use of the kinematics equations to determine the joint parameters that provide a desired position for each of the robot's end effectors. Arduino IDE is used for XYZ positioning of robotic arm.

Lets demonstrate one example to calculate the angles required to move and position a robotic arm with 3 DOF to a particular point in a 2 dimensional X and Y coordinate space using Inverse Kinematics.

Origin: (0,0) x=0 y=0 -or- v=0 w=0
Target: (10,7) x=10 y=7 -or- h=10 k=7

Shoulder: 6 inches (Makes a 6 inch radius, Origin Servo)
Elbow: 4 inches (Makes a 4 inch radius)
Wrist: 4 inches (Makes a 4 inch radius, Target Servo)

To calculate slope of a line:
y=mx+b
m=(k-w)/(h-v) m=(7-0)/(10-0)=7/10 7=(7/10)(10)+b 7=7+b b=0 Slope:
y=7/10x

To Calculate a point where a line intersects a circle:
(x-h)^2 + (y-k)^2=(radius)^2 radius=4
(x-10)^2 + (y-7)^2=16

Part 1: (x-10) * (x-10) use FOIL method
x^2 – 20x + 100

Part 2: (y-7) * (y-7) use FOIL method
y^2 – 14y + 49 substitute Slope for y…(7/10x)^2 – 14(7/10x) +49
49/100x^2 – 49/5x + 49

Combining both equations:
x^2-20x+100 + 49/100x^2 – 49/5x + 49=16

Solving…1.49x^2 – 29.8x +133=0 A=1.49 B= -29.8 C=133

Use the Quadratic equation to solve for x:
(- B +/- squareroot(B^2 – 4(A)(C))) / 2A = x

Use the A,B and C values from above to find x. There will be 2 answers. One for the plus (+) equation and one for the negative (-) equation.
x= 13.3 and x=6.7 (I only use the smaller number)
Use the Slope equation from above to solve for y… y=(7/10)(6.7) y=4.7

The point where the Slope crosses the circle is (6.7,4.7).

Use the Pythagorean theorem to calculate the distance from the origin (0,0) to the new point (6.7,4.7)…distance = squareroot(x^2 + y^2)

squareroot ((6.7)^2 + (4.7)^2)=8.18
With 3 known sides of a triangle use the Side-Side-Side calculation to find all 3 angles of the triangle: Cos(A)=(B^2+C^2-A^2)/2(B)(C)
Cos(B)=(C^2+A^2-B^2)/2(A)(C)
Cos(C)=(A^2+B^2-C^2)/2(A)(B)

A=4 (length of target servo)
B=8.18 (length of origin to new point)
C=6 (length of origin servo)

Cos(A)=66.91+36-16/98.16=0.8853 Angle A = 27.71degrees
Inv Cos(0.8853)= 27.71 degrees

Cos(B)=36+16-66.91/48= -0.3106 Angle B = 108.09 degrees
Inv Cos(-0.3106)=108.09 degrees

Angle C = 180 – Angle A – Angle B Angle C = 44.2 degrees
180-27.71-108.09=44.2 degrees

To calculate the full angle of angle A use the target x point as length of one side of the triangle and the target y point as the second length of the triangle. Use the Pythagorean theorem to calculate the third side length of (hypotenuse) the triangle.

Given 3 sides of a triangle, calculate the angle from the x axis to the Slope line using the Cos(A)=(B^2+C^2-A^2)/2(B)(C) formula.
Total Angle A = 27.71 + (angle from the x axis to the Slope line).

V. Hardware

4 DOF robot arm controlled by Arduino uno. The frame of the arm is 3d printed. All joints are module designed so you can easily change the number of joints. Main material is asbestos sheet. Along with bunch of nuts and bolts,small screws and 5 servos are used. Robotic Arms can be classified according to the number of "joints" or "Degrees Of Freedom" (DOF) they have.

FIG.5 Robotic Arm

i)The "Base" or "Waist" usually can turn the arm 180^0 or 360^0, depending of the Servo

type used. ii)The "Shoulder" is responsible for "raising or lowering" the arm vertically.

iii)The "Claw" or "Gripper" works by opening or closing to "grab things."

Servo motors mg996 and mg90 will be used to drive the joints, connected directly to the Arduino. The DC power for the servos should be separated from the Arduino and other components. An external power supply of 5 or 6V should be used. If the servos have problems and vibrate a lot, make adjustments at the "delays" of code. Also worth checking out if the servos are digital or analog, as though they are mechanically similar, the digital works with a frequency of 300Hz while the analog, 50Hz. The standard Arduino library "Servo.h" was developed for analog servos and can be modified if necessary, for a better functioning with digital servos.

VI. SERIALIZATION

Serial communication is used to transfer information between data processing equipment and peripherals. In serial communication, data is within the style of binary pulses. In other words, we can say Binary One represents a logic HIGH or 5 Volts, and zero represents a logic LOW or 0 Volts. Serial communication can take many forms depending on the type of transmission mode and data transfer. The transmission modes are classified as Simplex, Half Duplex, and Full Duplex. For each transmission mode there will be a source(sender) and destination(receiver).

While doing many implementations for embedded systems and their inter-device communications,we come accross 2 cases:
i) Sending data from PC to an actuator,like servo motor.
ii) Reading data from sensor like a switch and sending that to a PC.

This paper deals with "How to send data from PC to arduino using the USB communication".To illustrate this:
i) First make Arduino sketch on Arduino IDE.
ii) After uploading code to Arduino,we can start sending data from PC.
iii) On the PC side,with the help of Python script we will be sending data.
iv) To use the functionality of serial port in python,we need to install Pyserial on the PC.
v) After installing Pyserial,start writing python script in python shell.

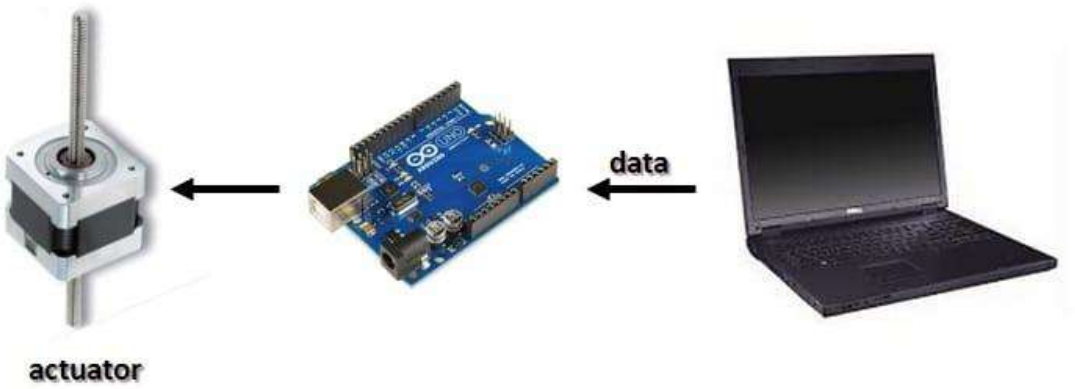

FIG.6 Serialization

VII. FUTURE SCOPE

A chess piece recognition may improve framework performance. Con to proposed approach false move detection requires a user interaction to correct misdetections. In the future, we intend to improve robustness of pro-posed system by considering better detection mechanism and improved performance in hardware and software, respectively. Building an Intelligent Chessboard for real time human vs AI experience with two-player remote gameplay using Mobile App is another future scope for our project. so basically setting up a game between two players with different geographical locations is intended in the future scope. It enables us to play our favourite game against a chess enthusiast from anywhere in the world. It can challenge the Artificial Intelligence of the board too. It helps us to detect illegal moves if human didn't place it correctly.

VIII. CONCLUSION

Chess remains a popular domain for experimenting in the field of robotics, computer vision and artificial intelligence. This paper gives an overview of the design and implementation of an Intelligent chess board; By the end we had accomplished all that we set out to do; build an intelligent chessboard that can play physical chess against a computer. The result was that it had the look and feel of playing against an actual opponent in real life rather than playing a computer game on the computer. We have aimed to apply the connected technologies such as the Internet of Things, Artificial Intelligence using Machine Learning algorithms and Image Processing. The game begins when a player makes a move then there is a part of Image Processing to identify configuration of the board using computer vision techniques it includes parts like capture chessboard image, image preprocessing, line and corner point detection,piece detection etc. This input is then given to the arduino for the processing then move generation using minimax /alpha-beta pruning. Reflection of that move on the board through robotic arm via serialization. Lastly ,the project is innovative as it brings new concept to the chess world, making people of different ages interested in enrolling into such a game.

IX. REFERENCES

[1]V.Wang,R.Green, "Chess move tracking using overhead RGB webcam," 2013 28th International Conference on Image and Vision Computing New Zealand (IVCNZ 2013), Wellington, 2013, pp. 299-304.

[2]VQ Nhat,G Lee, "Chessboard and Pieces Detection for Janggi Chess Playing Robot," International Journal of Contents, vol. 4,no. 4, pp. 16-21, Oct 2013.

[3]Doe-Hyung Lee,Kwang-Seok Hong, "Game interface using hand gesture recognition," 5th International Conference on Computer Sciences and Convergence Information Technology, Seoul,2010, pp. 1092-1097.

[4]David Urting,Yolande Berbers, "MarineBlue: A Low-Cost Chess Robot," IASTED International Conference Robotics and Applications, RA 2003, pp. 76-81, June 25-27, 2003, Salzburg, Austria.

[5]Sunandita Saker, "Wizard Chess: An Autonomous Chess Playing Robot," IEEE International WIE Conference on Electrical and Computer Engineering, p.p. 476 -478., December 2015.

[6]Brian,James, ”Automatic Chessboard ESE 350: Final Project,” Spring 2011.

[7]Chuck Orde, “Chess-Machine Learning,” 2018.[Online].Available: http://www.chuckorde.com/chess-vision-dnn-2.html