AUTONOMOUS CHESS BOT

Project- ROBOREX\_CHESS\_MATE

by Team ROBOREX

Team Members:

* Prabin Kumar Rath
* Neelam Mahapatro
* Subhum Sahoo
* Pabitra Dash
* Swostik Acharya
* Nischay Nirajit Mahanta

***INTRODUCTION: -***

Chess is an old and popular strategy game. We live today in an increasingly digitalised world and the technology has developed tremendously due to which computer chess games and online chess hubs have become very popular. Unfortunately they do not give the feeling of playing a real game of chess with chess board and pieces.

The objective of this project is to create an automated chess robot that will be able to interact with a human chess player and play full length matches. It uses available chess engines for predicting best possible moves, and one of such open source chess engines used in this project is STOCKFISH.

The game begins either with the user’s move or the stockfish’s according to the willingness of user. When user makes a move on the board, the current locations of the pieces are fed to the system after being detected by the camera. Stockfish gives a best move according to the resulting board configuration. The move performed by the stockfish is carried out physically on the chess board using an X-Y slider such that no two pieces collide with each other during the movements. An electromagnet is used to drag the pieces on board according to the commanded position changes from the system. User interfaces has been developed for interaction with the player and to give a real experience of chess in a more fascinating and interactive way.

In this project the focus is on sensing of the chessboard through camera feed and the autonomous movement of the chess pieces. The purpose of this project is to build a working prototype and establish a link between human player and computer chess engines in an interactive manner. Besides that, this game contains add on features of virtual chess games (New Game, Load Saved Game, Instruction Manual, Delete Saved Games, Restart on going Game, Undo Moves) to make Chess more interactive and interesting.

***GENERAL HARDWARE DESCRIPTION: -***

***Requirements: -***

1. 25’’ x 25’’ x 2.05” plywood base board

2. 24’’ x 24” glass plate

3. 17.6’’ x 17.6” flex of the chess board

4. 2 Railings on Y direction of length 24.63” and width 0.88”

5. 1 Railing on X direction of length 24.63’’ and width 0.88”

6. Racks of length 24’’

7. Plywood

8. 2 Stepper motors

9. Pinion of radius 1cm

10. Plywood level of 2’’ length and 0.75” width for placing electromagnet

11. Electromagnet

12. Camera to capture image of chess board

13. Black white strip for IR Feedback

14. 4 C pillars made of iron and steel

15. 32 chess pieces

16. Switch for confirmation of user move

17. Photodiode, High pass filters, IR LED for IR feedback

18. Fluorescent paper

19. C clamp stand of wood for fixing camera

At first, a baseboard was made of plywood keeping the dimension 25”x25”x2.05” on which the whole mechanical system was mounted. Two parallel Y railings were placed above the base which were of length 24.63” and width 0.88” on an aim for the safe movement of stepper motors. A rack of length 24” was placed near one of these railings and was fixed with the base board.

Above it, stage made of plywood of 24” and width 5.91” was settled in such a manner that it was exact perpendicular to the Y railings. Y stepper motor was mounted on the railings through a pinion of radius 1 cm by cutting the wood stage accurately. Above that wood stage, another Railing in X direction was fixed. Rack of length 24” was placed near the X railing and was fixed with the stage with an aim for the accurate movement of X stepper motor. Another plywood stage of dimension 9.84”x3.94”x0.38’’ was created for the mounting of X stepper motor.

The stepper motor was mounted on the slider via a mounting plate such that the gear of stepper motor mate with linear gear. X stepper motor was used for any movement of chess piece over X direction and similarly Y stepper motor was used for any movement of chess pieces over Y direction whereas the movement steps were determined by our code and finally fed to the motors.

Another wood level was constructed over the slider for placing the electromagnet. The dimension of this wood level was 2” height, 0.75”x0.75” area of cross section. Over this wood level, the electromagnet was mounted. Thus, the motion of the electromagnet was guided by the motion of the stepper motor. To avoid any positional error, we used two IR feedback system each of which consisted of a circuit containing photodiode, IR led wrapped with black tape and high pass filter placed with a support of small wood piece. This feedback system played a vital role in positioning the magnet in accurate position of the chess board. The black white strip was used such that the IR system can work with minimum error. We used two IR feedback circuits on a purpose to robust the motion of electromagnet in both perpendicular direction.

A square glass plate with each side 24” was fabricated, on which the printed chess board flex was attached. The flex had a dimension of 26.6”x26.6” such that each square box out of 64 boxes had a dimension of 2.20”x2.20”. Then the glass plate and the wood base board were settled together by 4 C pillars each of which had length 8.05”, width 2.79”. The materials for C pillars that we chose were iron and aluminium just to make the whole system strong and stable. 32 Chess pieces with fluorescent paper glued above each of them and neodymium magnets attached to the base of every chess piece were placed on the glass plate by observing the position on the flex.

A camera was fixed through a C-clamp at a sufficient height from the board such that it will be able to take images with appropriate resolution.

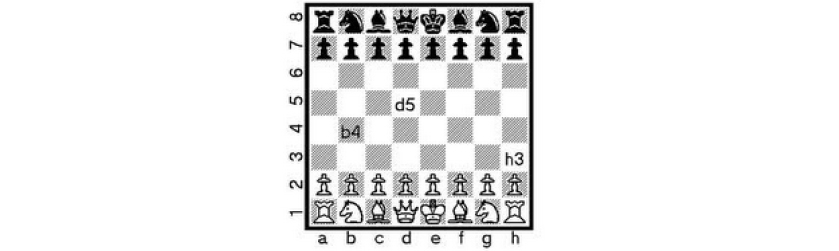


**Fig1:** **Chess Robot**

***SOFTWARE DESCRIPTION: -***

Software part of the project primarily uses well known applications and frameworks like Arduino, ROS, Qt and OpenCV. Basically, Arduino is an open source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board often referred to as a microcontroller and a piece of software or integrated development environment that runs on computer, used to write and upload computer code to the physical board. Arduino does not need a separate piece of hardware in order to load a new code on to the board it simply needs a USB cable. ROS stands for Robot Operating System which is a flexible framework for writing robot software. ROS is a collection of tools, libraries and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotics platforms. ROS is multilingual i.e. it supports development in multiple languages. ROS helps in establishment of communication between two or multiple program files each of which is written in same or different programming language. The sensor or actuator message from one program node is transferred to the other by a default or customised message file through ROS. Thus, ROS plays essential role in the intercommunication of different program nodes. The nodes we used basically in our projects here are namely chess\_ai(Python), board\_handler(Python), frame\_pub(CPP), frame\_handler(CPP), UI stack(CPP), Ardsimul(Python), Arduino code(Embedded C).

A general chess board is labelled in the diagram below. The positions are recognised by the computer algorithms according to this conversion as shown here.



Game is started from **chess\_ui** node. At the starting of game, a well framed user interface is displayed whose Menu bar has the buttons for New game, Saved Game List, Game Record, Instructions, and Credits. The tab Credits contains the developer list for the project. On clicking the Saved Game List, a list of all the games those had been played and saved without completion are displayed. User can delete any particular game which he played before without completion or could resume the game from where he left. Before beginning of game, the user is advised to refer the Instruction shown in instruction tab. On clicking the New Game option, user has to enter the name of event, name of white side player, name of black side player(default Stockfish for engine) and the round number. The player can choose white side or black side according to his/her willingness. After confirming these data, the player is directed to another UI which shows a live status of the chess board in graphical fashion and the turn of each side is represented by a dot (Stockfish or human player) thus initiating the game.

The movement accomplished by the user is detected by camera which captures the image of the whole chess board including the pieces. **Frame\_pub** node receives the camera image which is in RGB format and using OpenCV functions encodes the image to a sensor message and sends that image to **Frame\_handler** node through ROS.

**Frame\_handler** node, after obtaining the message, decodes it to get the image. This code has functions to crop the image separating all the disturbances outside the chess board those were captured by the camera. Another function is called to convert the obtained cropped image to HSV format. The HSV image is converted to binary image (suppose image 1) by colour segmentation methods to detect the fluorescent yellow paper attached to the head of each chess pieces. Another copy of cropped RGB image is also converted to grayscale image. The grayscale image is blurred for a purpose to apply an edge detector to find edges of white squares. After finding the edge of white squares, mathematical operations are performed to find the centres of all these 32 white squares and those coordinates are sorted. Applying abstract operations all 32 centres of black squares are also detected. After this operation we have an image (image 2) with marked centres of all the centres of 64 squares. Image 1 and image 2 are combined to acquire all the positions where chess pieces are present on board. After formulating all the positions, it is encoded into a string format and is sent to the **board\_handler** node through a message file.

The published binary string from **frame\_handler** node is received by the **board\_handler** node. After confirmation of the user move (confirmed by player when he pressed the switch), the board handler node has function to generate a new move string by comparing the changes in two concurrent binary strings.

Example: - For b5 to d5 move the string would be like (b5\_10, d5\_01). Similar strings are sent to the **chess\_ai** node after each user move confirmations.

Diagram: -

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | h |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | g |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | f |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | e |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | c |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | b |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | a |

1 2 3 4 5 6 7 8

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | h |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | g |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | f |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | e |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | d |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | c |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | b |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | a |

1 2 3 4 5 6 7 8

**Chess\_ai** node acquires the string from board handler node and converts it into UCI form. UCI is Universal Chess Interface which is a protocol for communicating with the chess engines, here to communicate with stockfish or similar engines which use UCI. The UCI form of the above move is simply b5d5. This code has a function to check first whether the move accomplished by the user was a legal move or illegal. If the output comes illegal, then a warning message is sent to User interface to warn the player about the illegal pass and the user is supposed to correct the illegal move which would be updated in **board\_handler** node and now it again comes to user’s turn to accomplish a legal move. If the movement comes out as legal then the UCI string is transferred to the ASCII matrix. ASCII matrix is internally stored from which STOCKFISH observes the user’s matrix and generates a best possible move according to the updated board configuration. The move given by the STOCKFISH is updated in the ASCII matrix and it is also updated in the **board\_handler** node such that in the next step the user’s pass would be compared with the updated matrix. ASCII matrix has the same configuration as that of physical chess board after every move. The current FEN is frequently passed to the user interface and updates the graphical UI with the current board configuration.

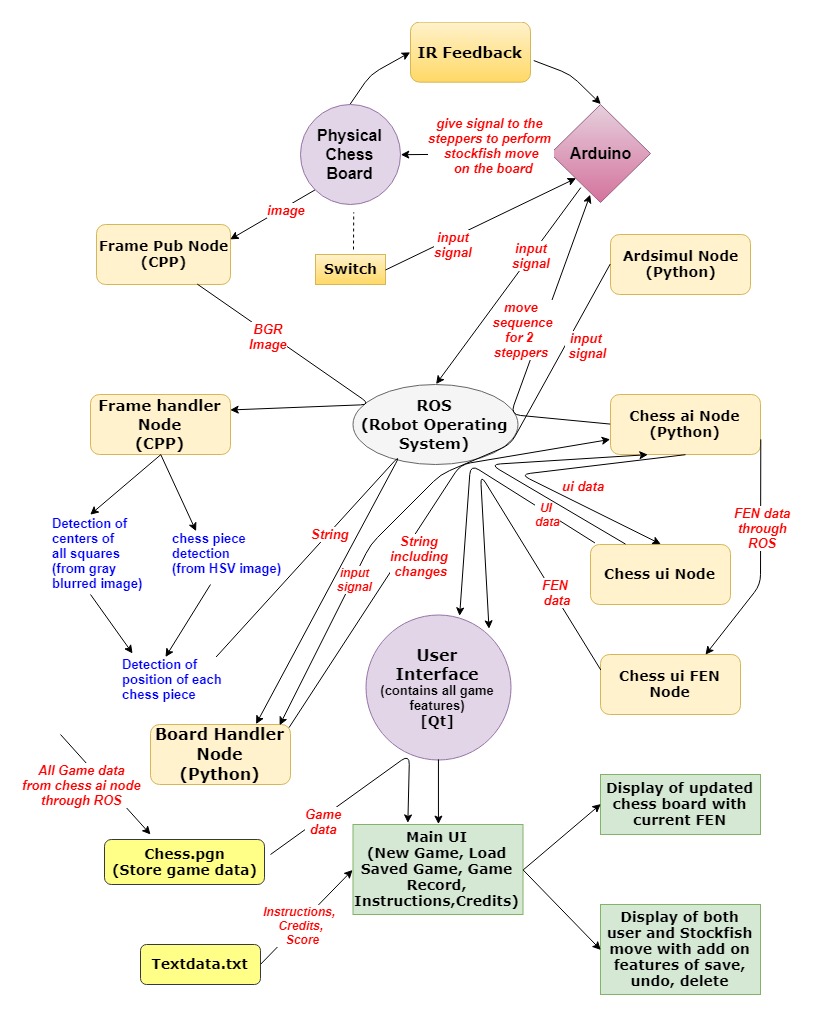
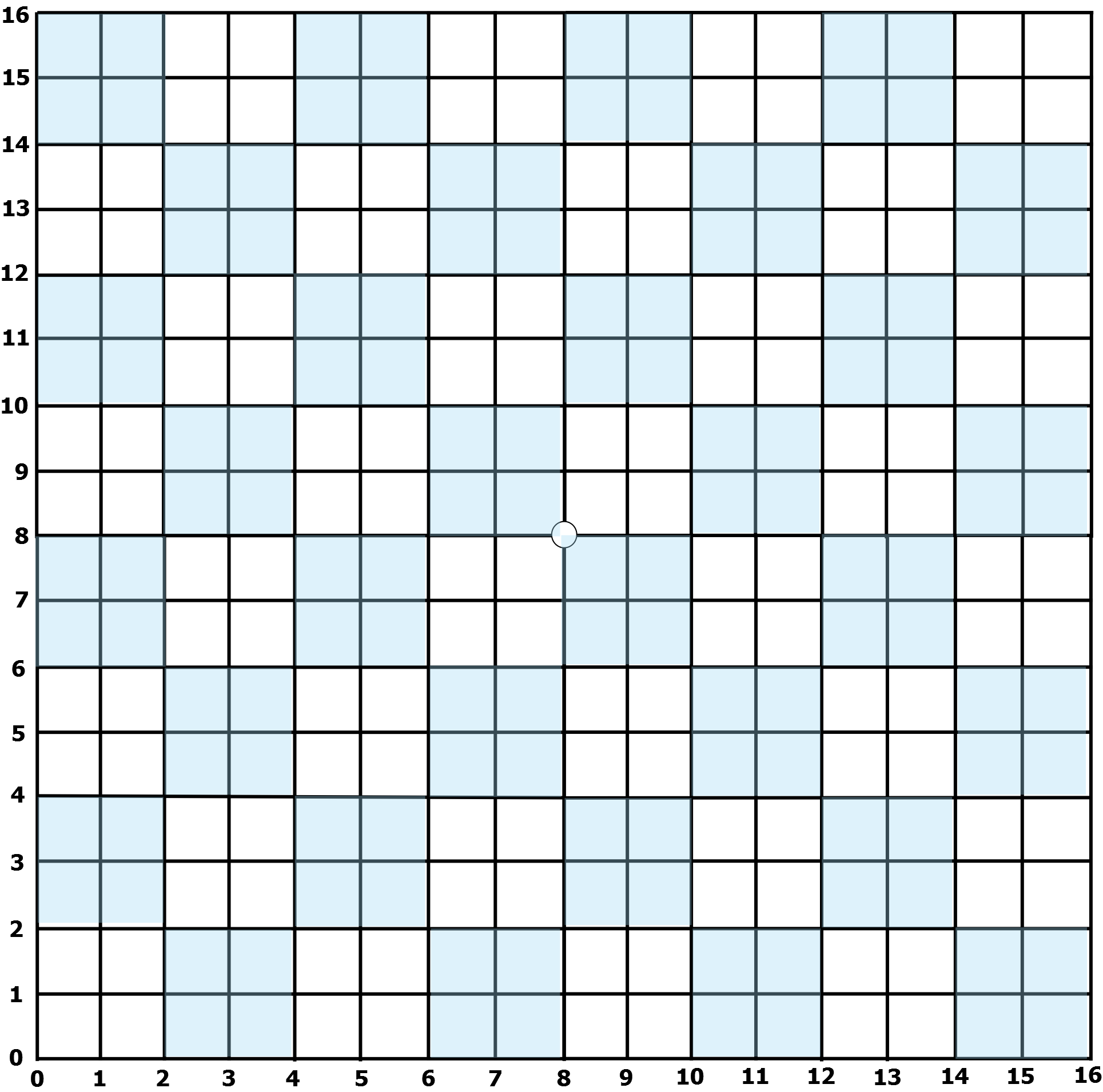
The most important part, the output move given by the engine has to be performed physically on the chess board for which a reference 16X16 matrix is used to generate a move sequence, and accordingly signals to the stepper motors are sent for moving the piece on the board with the help of electromagnet which further involves shortest path and encoding and decoding algorithms. The electromagnet proceeds towards the initial position of targeted chess piece, turns on, picks up the chess piece and places it in the specified position under the guidance of IR feedback system. The **UI Stack** contains a collection of UI files and codes. During the game, every movement is updated in UI. There are also additional functionality like save, undo, delete, loading of saved game and crash management to provide features of virtual chess games as available in Market. All the codes for User Interface were written using **Qt** which is a cross-platform end user application development framework and widget toolkit for creating classic and embedded graphical user interfaces. 

Fig2: The above diagram shows a sample flow chart how this project functions.

***XY TABLE SOFTWARE DESCRIPTION: -***

It’s a 16X16 matrix. The 8X8 matrix was transformed into 16X16 matrix for moving the pieces by the help of electromagnet on the chess board automatically in a collision free manner. So, there was a need of co-ordinate conversion between these 2 matrices. We designed the XY magnet system so that it could be able to transport the pieces along the lines in between the squares so that the pieces does not collide. This was the reason why the board squares needed to be a bit large. XY magnet system needed to have a finer coordinate system so that it could both go to the location of the piece and pass in between other pieces. The following grid shows the coordinate system of the XY magnet system. The coordinates here range from 0 to 16 in both X and Y directions.

The transformed numerical move coordinates would again need to be changed to fit the granularity of the XY system coordinate system. This involved multiplying each coordinate by two and adding 1: (2x+1,2y+1). Thus, if the instructions were to move piece at (6,2) to space (5,2) received, this would correspond to the XY system needing to move the piece at (13,5) to (11, 5).

**

*Fig3: 16X16 Matrix*

The electromagnet used to start from the centre of the board for each Stockfish move, which is positioned at (8,8) in XY System and returned to the same position after performing a particular movement of chess piece. This was performed with an aim to maintain a constant reference point and to avoid any mechanical error that could have possibly generated. After perfection of all mechanical components, we still got some mechanical error in the positioning the piece due to which we decided to work with IR feedback system which helped us to further perfect the output.

So, when the origin coordinates were received, we subtract (8,8) from the coordinates. For example, (13,5) – (8,8) = (5,-3). The Y stepper motor then had to shift down five coordinate steps and X stepper motor then had to shift to Left 3 coordinate steps. After the XY magnet positioner arrived to the initial coordinates, the electromagnet was then switched on to attract the piece. Then the current coordinates were subtracted from the next coordinates. For example, (11,5) – (13,5) = (2,0). Thus, the Y stepper motor needed to shift down 2 coordinate steps. The X stepper motor didn’t need to shift in this case. However, to avoid piece collision we wanted to travel along the lines instead of through the squares. For that reason, we used the concept of offset here. Thus, each time we are moving a piece, we first shift up and right 1 coordinate step. So, we’d go from (13,5) to (14,6). Then we would do the (-2,0) shift, thus going from (14,6) to (12,6). Then we would cancel out the offset shift by going down and left one coordinate, thus going from (12,6) to (11,5). The piece then arrived at the destination coordinates.

According to this method, the move sequence was generated which was used to determine the number of steps needed to be moved by both the stepper motors separately. The **chess\_ai** node had the function to determine these move sequences and accordingly passed to the Arduino to guide the movement of stepper motor.

This then completes the construction of the XY table. The final part of construction on the XY table involved wiring up the stepper motors. A motor driver was needed for each stepper motor.

***IR Feedback: -***

The main purpose of using IR Feedback system was to avoid the mechanical error generated while positioning the electromagnet as per the move produced by stockfish for a certain chess piece. We had used LM 324, a quad opamp for our feedback system. This system is based on the working principle of photodiode and IR led. The pattern we used consisted of alternative black and white strips each of 3mm and a total of 164 strips placed near both the X and Y slider. When IR pair passed over a white strip the output voltage of the circuit raised to 5V and this voltage fell to 0V when the strip was black.

An interrupt function was constructed to detect the voltage spikes and thus it tracked the position of the electromagnet. More over to say, this system was built to inspect whether the magnet had moved to its specified position or not. If not, a feedback would be given back to the system which would suggest further steps to the motor, and this feedback would work unless and until the magnet reach its specified position.

***Electronics Description: -***

***Requirements: -***

1. Arduino UNO

2. ULN 2003 - 2no.

3. N-channel MOSFET

4. LM 324

5. Red LED

6. Green LED

7. IR Pair-2

8. 1uF capacitor-2

9. 0.1uf capacitor-2

10. IN 5408

11. IN 4007

12. LM 2596

13. 330 ohm resistor-4

14. 10k ohm resistor-4

15. LM 7805

16. Potentiometer (100k ohm)-2

17. HC05-Bluetooth Module

18. ESP8266 Wi-Fi Module

We supplied 12 volts from a SMPS source which was connected in parallel to buck converter, LM 7505, Arduino UNO and the electromagnet. Buck converter provides a voltage output which varies from 0 to 12v with a high current. In our case we used it to get a 5v high current output. This 5v output was connected in parallel with two ULN 2003A, with 0.1uF capacitors connected across each of them. The output pins of these ULN2003A were input to the stepper motor along with 5volt supply from buck converter. Stepper motor requires a high current to work which was a reason to use Darlington transistor ICs(ULN 2003A) here.

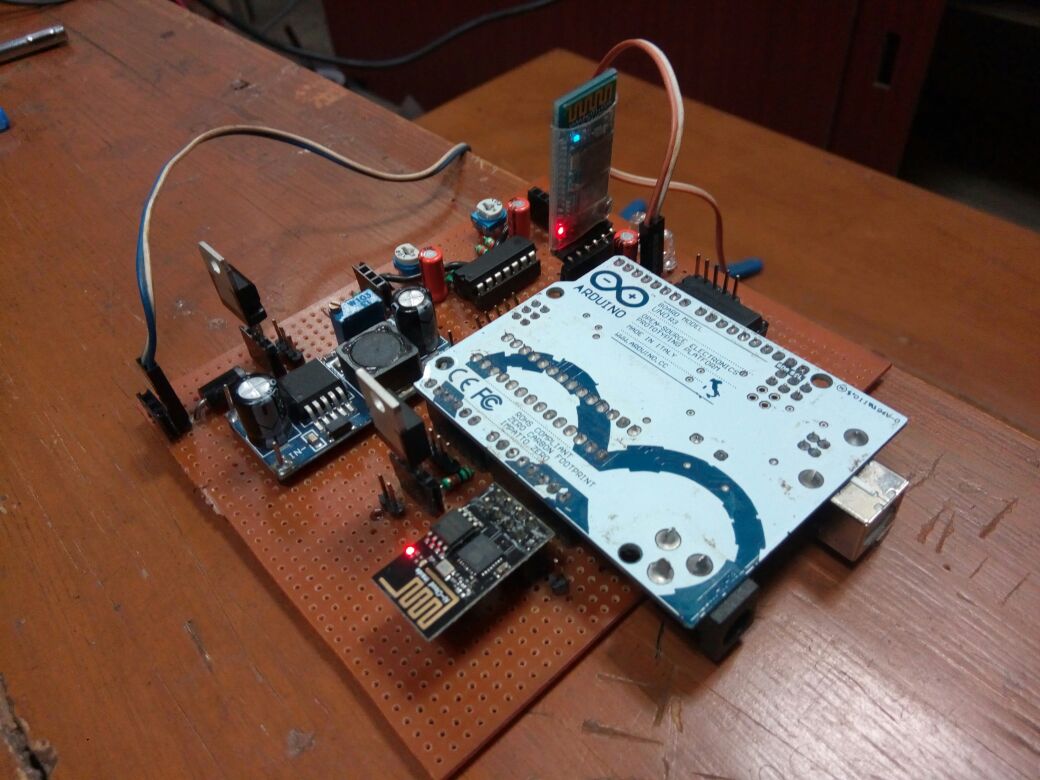


Fig4: **Final module (Electronics Circuit)**

The output of LM7805, a 5v voltage regulator was provided as power input to LM324, HC05 and IR Pair. 1uF capacitor and 10k ohm resistor were used for the high pass filter for allowing only the low frequency signals from the photodiode and switches to be received by Arduino.

The electromagnet was supplied with 12 volts. N-MOS connected with 10k ohm gate resistance was used as switch for the electromagnet. The switch was controlled from Arduino according to the commands received from the system.

***Conclusion: -***

All in all, the project has been made to give the user a realistic experience of playing Chess. The dots move by themselves with many tasks going on in the backend. The biggest challenge was to create a fully functional and working interface between the physical world and the engine. The project really stood up to the mark. We are still working on the project keeping in mind the new changes that can be appended to make it’s working more robust. It was really a great project to work on and the dedication and perseverance shown by the team members really paid off making the project a success.

*Git hub link of our Project: -*

[*https://github.com/prabinrath/Roborex\_Chess*](https://github.com/prabinrath/Roborex_Chess)

***References: -***

1. <https://python-chess.readthedocs.io/en/latest/> Python-chess is an open source python module used for handling chess engines.
2. <https://stockfishchess.org/> An open source chess engine we used as default engine in our project.
3. <http://wiki.ros.org/> ROS official Tutorials.
4. https://www.dribin.org/dave/keyboard/one\_html/ Similar projects.
5. <http://www.instructables.com/> DIY community for electronics projects.
6. <https://playground.arduino.cc/> Arduino Official forums and tutorials.
7. <https://www.qt.io/> Official Documentation for using Qt with C++.
8. <https://opencv.org/> Open source Computer Vision Library for image processing.