Intelligent Chess Robot Tutor: A Low-Cost, Vision-Guided Multi-Level System with Real-Time Tracking and Advanced Gameplay Analysis

1st Iyed Mdimegh
INSAT
Tunis, Tunisia
iyed.mdimegh@ieee.org

2nd Mohamed Hamzaoui
INSAT
Tunis, Tunisia
Mohamed.hamzaoui@ieee.org

3rd Malak Bouzidi
INSAT
Tunis, Tunisia
malakbouzidi@ieee.org

4th Rayen Kasmi
INSAT
Tunis, Tunisia
rayenkasmi@ieee.org

5th Ahmed Zghibi
INSAT
Tunis, Tunisia
ahmed.zghibi@ieee.org

6th Fayez Zouari
INSAT
Tunis, Tunisia
Faayezzouari@ieee.org

***Abstract*—Chess is a highly popular board game that is to this day practiced between individuals of different ages despite the rise of digital games. To bridge the gap between traditional and modern learning methods, this paper presents RoboKnight, a multi-level system that focuses on teaching chess to individuals regardless of their level using a scalable, modular architecture. The system employs an affordable 4-DOF robotic arm equipped with a robust dual-power electronic structure to enable reliable and secure performance. The multi-stage computer vision pipeline with YOLO models and FEN generation supports accurate chessboard and pieces identification and interpretation for real-time move tracking and validation. A central logic unit governs the integration of computer vision, robot control, and chess engine to control game flow and chess rule enforcement. Finally, a web-based dashboard enables real-time monitoring, remote operation, and performance tracking of multiple Robot arms, enhancing the educational value of the system. Although the present prototype can deliver an interactive and functional learning experience, it is restricted by the absence of multi-device synchronism and conversational feedback. The envisioned upgrade aims to incorporate interactive coaching via conversation with a chatbot, a single Raspberry Pi coordinating multiple arms via task scheduling, and power efficiency via dual-battery switch with solar recharging.**

***Index Terms*—Robotic Chess Coach, Educational Robotics, Artificial Intelligence in Chess, Computer Vision, Adaptive Chess Engines, Human-Robot Interaction, Robotic Manipulation**

I. Introduction

Board games like chess have never failed to provide an open field to test problem-solving skills and strategic planning for all age groups [1]. Changing from a game practiced by the noble few to the most popular board game worldwide, chess has been growing further thanks to the development of advanced algorithms and artificial intelligence [2], [3]. However, despite their informative experience, digital chess games have moved away from the tactile nature, creating a gap between virtual platforms and over-the-board play [4].

Moreover, the traditional methods to learn chess are limited to three approaches: human tutoring, software-assisted learning, and self-studying using books. While human tutoring is a great way to get active feedback and a custom teaching process, it is usually less available and significantly more expensive. Learning through chess software also offers further analytics but lacks the tactile, board-based experience. Self-education using books is the most accessible but lacks the active feedback we get from other approaches.

As a result, the gap can be filled by making robot solutions more interactive and informative in a personalizing way. Several robot systems in the past attempted to automate chess using robotic arms and computer vision [5], [6], [7], [8]. Yet, all of these systems biased gameplay function at the expense of educational value. For instance, although Gambit [6] and the Baxter human-like chess player [5] demonstrate high levels of automation, they do not utilize an educational layer. Similarly, attempts such as [7] and [9] focused solely on move prediction or control of arms with low levels of preprocessing and no player adaptation, but without offeringplayer guidance or accommodating the skill level of the player. Additionally, such systems are often founded on costly components or require specially crafted pieces to aid in manipulation as well as in vision tasks, so they are unaffordable in public school settings [5], [8]. RoboKnight, by contrast, aims to fulfill the gap by offering a low-cost, transportable robot chess tutor that can play chess, as well as serve as an intelligent teaching aid. RoboKnight is founded on commercially available materials, does not require special pieces, and keeps power costs low with the provision of solar power, batteries, or AC-DC adapters.

The RoboKnight structure can be divided into four main parts: First, computer vision is applied to detect the chessboard and pieces. Second, after making sure the image is processed correctly, an advanced chess engine is implemented to predict the best move and develop other game modes. Third, the robot arm handles the maneuver of chess pieces. Finally, the chess coach reports the detected moves to a web page in order to

get more insights about the game.

The key value of this project is further extending the capability of the present chess engines and blending it with physical demonstration. RoboKnight will not only analyze the played moves in real time through a user-friendly interface, but also adapt to the player's level to give them challenges adjusted to their rating. As a result, we end up with a mixture of physical demonstration and advanced chess coaching capabilities in order to achieve a better learning experience.

In the following sections, we detail the design and implementation of RoboKnight. Section II presents the system architecture, outlining the hardware and control structure. Section III discusses the 3D design of the robotic arm, followed by Section IV which explains the electronic setup. Section V elaborates on the computer vision pipeline, while Section VI describes the main logic of game interaction. Section VII introduces the learning dashboard that enhances user interaction, and Section VIII evaluates the system's performance. Finally, Section IX explores potential improvements and future enhancements for the system.

II. SYSTEM ARCHITECTURE

Three main parts come into play to deliver a personalized and dynamic learning experience. The first part is a mini-computer, the Raspberry Pi. It's the master or the brain of the system and its high-level processing unit. Second is an Arduino Uno playing the follower role. This part is responsible for the low-level control which is executing its master's orders to control the movement of the arm [10]. Third, we have a web server where the system saves all the games for more insights. We opted for a master-follower architecture for modularity reasons. Such architecture makes control easier and eliminates collisions and inefficiencies since the master is the only part accountable for making all the decisions and giving all the orders. In addition to its efficiency, the system becomes far more scalable. As the Raspberry Pi won't be responsible for carrying out the actual tasks, there will be a lower load on it, hence more room to add scalable features. We can also upgrade our followers and even integrate more followers without having to modify the whole system.

To fully explain the system architecture, shown in Figure 1, in a simpler way, we will break it down into steps. The first step is setting the gameplay and the mode. Our user interface is composed of three buttons and an LCD. The first two buttons are used to navigate and select the game mode and the ELO, a metric measuring the skills of the player based on results of past games. Meanwhile the third button is used during the game to ask the robot for assistance and move recommendations. The system will interact with the player through the LCD, sending alerts, insights, and feedback.

Our system has the ability to provide 3 different gaming modes, each of them having its own educational prospects. The

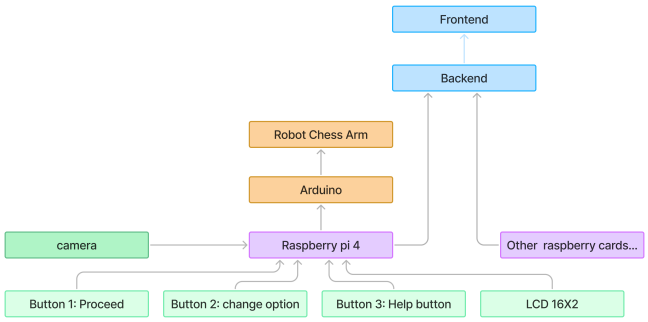

Fig. 1. Over all design of the robot arm

first mode is a classical mode in which players are expected to participate in a simple game against the robot. The second is the educational mode. Its similar to the first one except that it possesses the possibility to allow the player to get some move suggestions by its rival the robot. Finally the last one, the puzzle mode, which is more challenging than the first two as it puts the player in a sequence of the game where he should figure out the possible moves. This mode aims to develop tactical and strategical skills [11].

After setting the game, the opponents take turns playing while a camera captures the board frequently and updates the game's state. The recurring frame capturing may cause system overheating and a huge energy consumption; thus, to reduce such side effects, the camera captures frames only when the player presses the validation button after finishing his move. A computer vision code and AI models process the captured frame in order to extract the board status. Afterwards, the system inspects the validity of the player's move and then moves to chess logic to pick the best move for the robot.

Rather than developing a chess engine from scratch, we chose to use Stockfish, the strongest open-source chess engine. With the use of Pystockfish, an open-source Python wrapper, we were able to fully integrate the chess engine into our program, which allowed us to customize player color and difficulty and determine the best move for a given chess position [12]. We additionally used the chess Python library to verify the legality of the moves made by the player [13].

Once the robot decides which move to play, it's time to carry it out physically. This mission is performed by the Arduino Uno. Using a mapping algorithm, the Raspberry Pi determines the full coordinates of the move and sends them as an order to the Arduino via serial communication protocols [14]. Through PWM signals, the follower controls five servomotors, each responsible for a joint in the robotic arm, to realize the move.

Finally, we arrive at the last component of the system. Our centralized dashboard provides tracking of the whole system. It offers system monitoring and real-time game tracking. Moreover, it observes the status of robotic arms in many units. To put it simply, the dashboard delivers a full performance analysis for multiple robotic units, widening the learning

experience beyond individual gaming sessions.

III. 3D DESIGN

The robotic arm is designed in Solidworks to meet our needs in moving chess pieces, adopting methodologies inspired by recent advancements in low-cost 3D-printed robotic systems[10]. We focused on three key areas: the degrees of freedom, the gripper's strength, and the base stability, to serve our objective precisely, as shown in Fig.2. Similar to the 4-DOF rover arm described in [11], we created a structure with 4 degrees of freedom prioritizing modularity and reduced mass, which is sufficient to place a chess piece anywhere on the board. This includes four joints and a servo motor in each: base, shoulder, elbow, and wrist. By combining the degrees of freedom and the arm dimensions, which are 4 cm base to shoulder, 25 cm shoulder to elbow, and 22 cm elbow to wrist, we had an efficient working envelope for our robotic arm with a maximum range of 51 cm that meets the standard dimension of a home size chessboard. To enhance the precision and responsiveness of the mechanism, the shoulder is attached to the base with a mechanical play of 5 mm for free rotation. For base stability, we attached a counterweight to prevent tipping during operation.

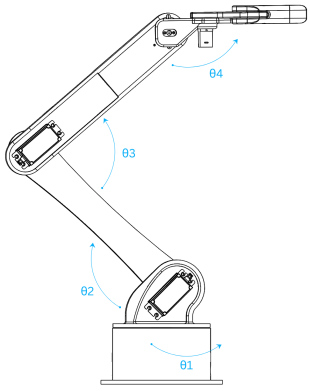

Fig. 2. Robotic Arm articulations.

The gripper, shown in fig 3, was created with a special form to hold tightly the different chess pieces without touching the others on the board. The jaws with 10cm length were designed to meet standard chess piece dimensions and were covered with a thick layer of foam, allowing successful grips with the friction needed to prevent any sliding of the piece for more precision.

While the authors in [11] utilized aluminum tubular links and 3D-printed joints for planetary exploration, our arm employs PLA-printed components with a 20% infill, ensuring a light weight approach while accommodating the specifications of our servo motors which are three MG995 servos and two MG90S servos.

IV. ELECTRONIC SETUP

The wiring shown in Fig.4 and Fig.5 is the full build of the robotic arm project, including the battery charging

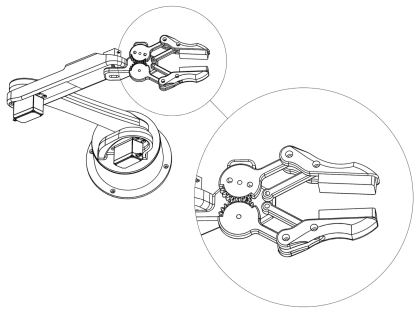

Fig. 3. Gripper Design.

system, the controller unit, and the actuators.The input power supply that our system requires is 15V DC with a maximum 4A current demand. These specifications offer the optimal working conditions for the robotic arm since the current draw and the voltage usage of every electric component are as follows: for the three high-torque servos[12],500 mA to 900 mA (7V), and for the two low-torque servos,120 mA to 250 mA. Our system uses a multi-step voltage regulation structure to support higher power loads, whereas previous like [13], utilized Arduino-based devices that had simpler power structures to power servos. We downgrade the 15V input voltage to supply each component's voltage needs through the use of DC-DC converters.

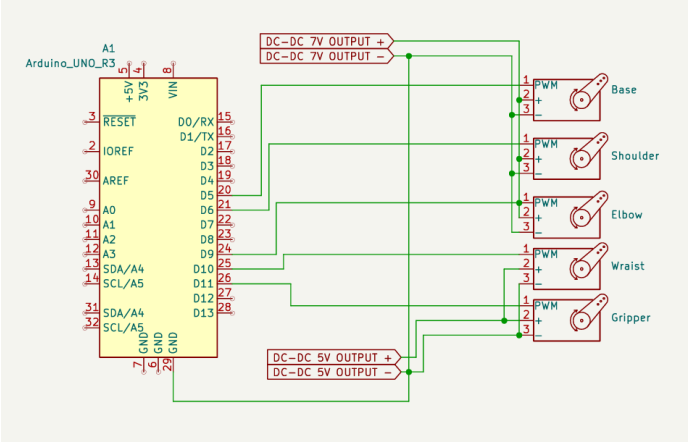

Fig. 4. Wiring Diagram.

When unplugged from the DC power source coming from the AC/DC converter (220V to –15V), the system works properly with the three Li-ion battery cells (3.7 V each) to power the actuators and the power bank for the Raspberry Pi, which in its tour powers the Arduino Uno and the command board. The separation between power sources ensures the safety of the Raspberry Pi since the sudden current variations caused by the servos can damage it. Now, by plugging in the DC source, the PNP MOSFET switches from the battery source to the plugged source, and the battery charging system will work instantly, so we will get two parallel functionalities: the operation of the robotic arm and the charging of the batteries.

This switching mechanism was inspired by the configuration proposed in [14], where the use of dual power sources is suggested to prolong battery life and ensure system continuity.

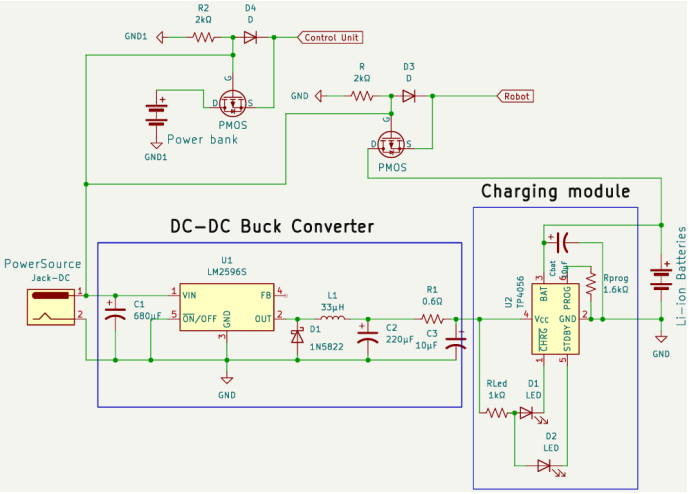

Fig. 5. Power source circuit.

The battery charging process is typified by a high number of challenges, including both overcharging and overheating complications. As a result, within our battery charger development, we utilized the LM2596 DC-DC buck converter module to maintain a steady input of 5V to the TP4056 charging module, which is arranged in parallel for every individual battery. In light of the results presented in [15], the use of the TP4056 as a charging module has been shown to be a viable and economic substitute for sophisticated charging management systems, providing key protective features, including over-discharge and short-circuit protections, without a specialized control unit.

V. COMPUTER VISION SYSTEM

The integration of computer vision is one of the core components of RoboKnight. It allows the robotic arm to "see" the chessboard and react according to the positioning of the chess pieces using captured images as input. However, such raw entry needs multiple preprocessing steps before it is ready to be handled. In addition, after processing it, it needs to be put in a convenient format for the game logic program to understand. That's why, we implemented a multistage process including image capture, chessboard detection, perspective transformation, grid mapping, square localisation, chess piece recognition, FEN generation, and Chess Move Detection and Validation.

Initially, the captured image needs to portray the whole chessboard while guaranteeing that the chess pieces are distinguishable and do not overlap each other or cover one of the corners. Therefore, after testing multiple angles, we positioned the camera at 60-degree angle relative to the chessboard. Next, the captured 640×480-pixel image is fed into our preprocessing module.

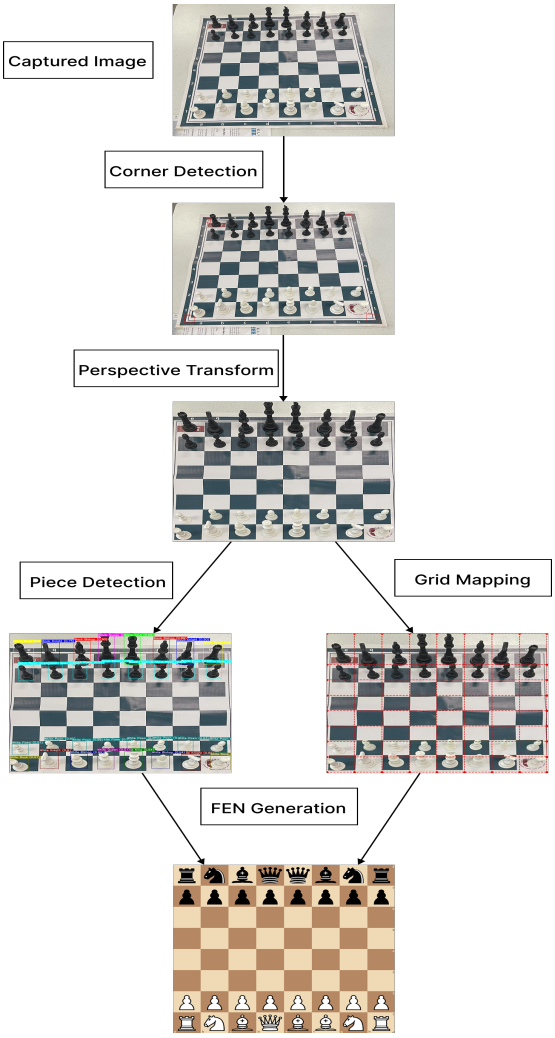

Fig. 6. Computer vision algorithm flowchart with intermediate screen shots at each stage.

Subsequently, in order to detect the chessboard, we used a YOLOv11 model (You Only Look Once) and fine-tuned it on over 10,000 images to reach 90% accuracy for corner detection [21]. The pictures were fetched from Roboflow, a platform that allows data annotation, preprocessing, and model training. The model detects the corners, assigns to each one of them a bounding box and a confidence score, picks the four corners with the highest scores, and then returns the center coordinates of the bounding boxes.

Afterward, we apply perspective transformation in which we start by ordering the center coordinates in the clockwise direction from the bottom-left corner. We subsequently apply a four-point perspective transform to correct perspective distortions and to reduce background noise, isolating the chessboard in the image [22], [23]. However, this process usually crops off the top-row pieces, that's why we add margin to the positions of the top two corners, resulting in a better view over the pieces. In the end, we get an image where the chessboard squares exactly coincide with the vertical and horizontal axes.

Following this, we divide the transformed image into an

8x8 matrix. Four corner coordinates of a square on the actual chessboard are represented by each value in the matrix. This step converts the visible chessboard to a coordinate matrix, making exact piece to square mapping possible in the later steps.

Next, for the chess piece detection, we used a YOLOv8 model and then further fine-tuned it on over 20,000 images on Roboflow attaining a 95% accuracy [24]. Using the provided transformed image, the model detects the chess pieces, labeling each one of them with a bounding box and a class that represents the type (pawn, bishop, knight, rook, queen, king) and color (black, white) of the piece. We noticed that bounding boxes for tall objects like the king and queen cover 2 adjacent squares in the board image. To prevent this, we added a height limit for the boxes so that each box covers just one square on the board.

Finally, for FEN generation, for each piece found, we compute the intersection over union (IoU) of its bounding box with every one of the 64 board squares that we have precomputed [25]. The board square where the chess piece lies is the one that has the maximum IoU and is therefore utilized to determine the rank and file of the chess piece. These positions are then systematically transformed to FEN representation.

In order to know whether a player has moved, we compare the found FEN with the last correct FEN, as these FEN representations should be equal or differ by one move. Any abnormal detections or sudden board state changes are then marked for re-evaluation [12].

VI. Main Logic

The general logic of the process of the chess robotic arm includes the interaction between computer vision, robotic arm operation, and chess engine reaction. The process involves three steps: initializing the game, making moves, and controlling the state of the pieces.

A. Initialization Logic

The system initialization section initializes the operational components needed for the following steps of the program to operate. First, we initialize the GPIO pins to run the arm. Next, we establish serial communication between the Raspberry Pi and the Arduino.

After that, we define the confidence threshold for the computer vision model. We then extract the chess square mapping on an 8x8 matrix to real-world coordinates. Next, we adjust game parameters based on the opponent's ELO rating, game type (classical or tutorial), and allow the player to choose a color. Finally, we initialize Stockfish based on the player's setup.

B. Robot's Turn Logic

On the turn of the robot, the program employs the Stockfish chess engine to determine the best move, translates the resulting piece location into a physical move, and moves it on the chessboard in a tailored way by issuing a command message. Specifically, we put the message in a "X1:Y1—X2:Y2" format to indicate the source point for the move "X1:Y1" and give the destination point "X2:Y2", where X and Y represent respectively the x-coordinate and the y-coordinate of the chessboard.

The message is then transmitted to the Arduino via the serial communication created. The robotic arm is moved using inverse kinematics to translate actuator movement (servo motors) into Cartesian coordinates.

C. Opponent's Turn Logic

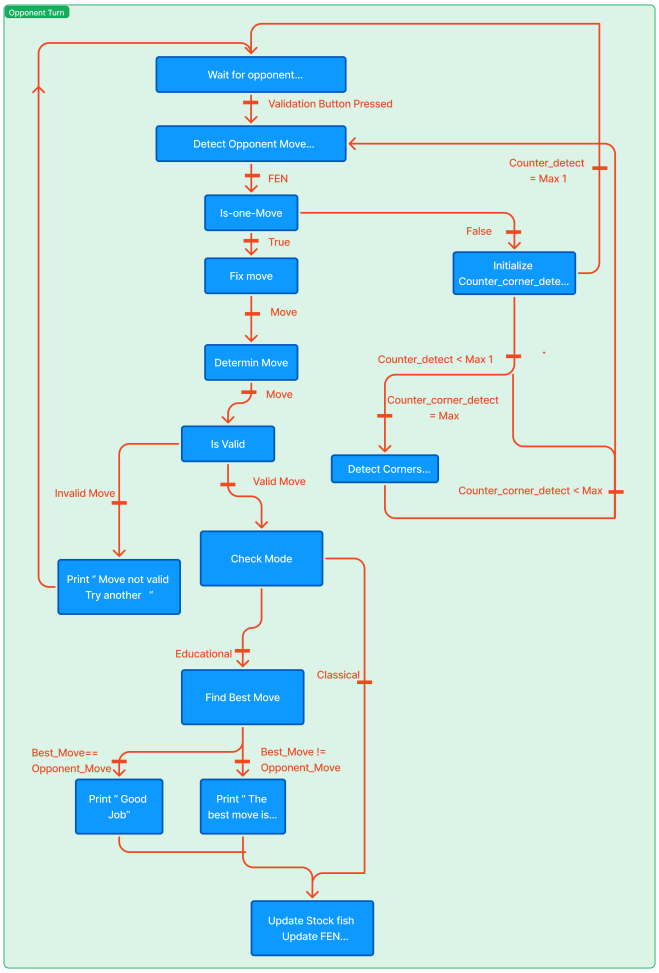

Fig. 7. Opponent move detection and validation workflow.

When it is the opponent's turn, the program takes a picture of the board to evaluate the game state, renders a FEN (Forsyth-Edwards Notation) representation, and validates the opponent's move to check its legality [120].

If in learning mode, the program compares the opponent's move with recommendations from the Stockfish chess engine and provides feedback.

D. Main Logic

The primary program orchestrates the overall flow of logic and utilizes the function `main()` to facilitate toggling between opponent and bot turns.

When designed with excellent error handling, the program allows for an endless loop until the game concludes or a human chooses to intervene.

VII. Learning Dashboard

The system includes an integrated, web-based dashboard that is used as an integrated monitoring and control system for involved robot arms used in chess education. It is accessed by all robot arms in secure HTTPS requests yielding live move information for analysis and tracking. The setup makes possible live game progression to be presented, complete histories to be stored, and in-depth analysis to be offered for post-match analysis. Remote control, live status reporting, and performance metrics are implemented in the interface, making the system especially well-suited to command many robots and students in unison. The use of robot arms in integrated digital platforms significantly boosts educational value in this system by enabling an interactive, hands-on experience through insights based in data.

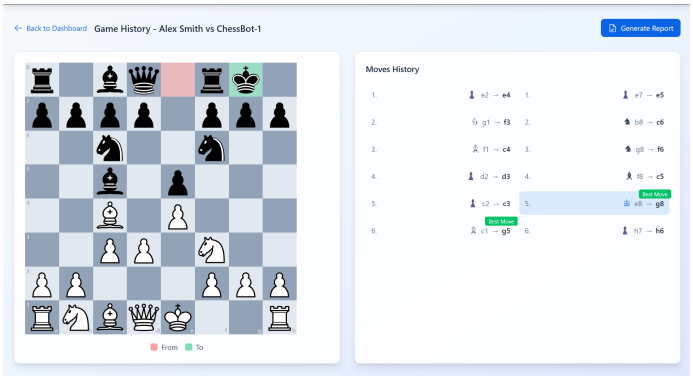

Fig. 8. Game history page

VIII. Results

The performance of our detection models is shown in Table I for YOLOv11 corner detection and Table II for YOLOv8 piece detection.

TABLE I
YOLOv11 corner detection model Results

Precision	Recall	mAP50	mAP50-95
0.89	0.78	0.86	0.53

The corner model achieves 0.89 precision and 0.78 recall with high localization accuracy of chessboard corners under varied viewpoint. The 0.53 mAP50-95 indicates the inherent difficulty in accurate corner detection with varied board viewpoints. Perspective transformations are applied in the following phases to improve overall board alignment and ensure consistency in the following processing stages.

TABLE II
YOLOv8 piece detection model Results

Precision	Recall	mAP50	mAP50-95
0.94	0.87	0.9	0.69

The piece detection model displays high recall (0.87) and precision (0.94) with low false positives and high detection across all types of pieces. The high mAP50 (0.9) shows its performance in ideal conditions, and the mAP50-95 (0.69) shows high overall robustness in less ideal conditions such as piece overlap or occlusions, a sign of well-generalized performance with very limited areas left to improve on.

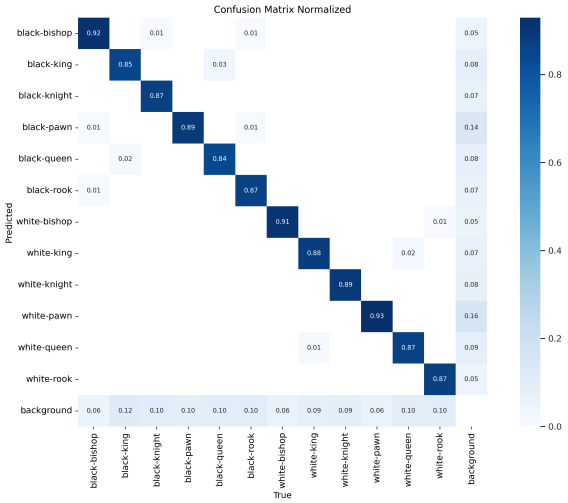

Fig. 9. Normalized confusion matrix for chess piece recognition. The model achieved 100% accuracy for the recognition of piece color (white/black) and 94% for piece type recognition..

Our model for chess pieces is highly accurate with very little possibility of missclassifying the types of pieces. While a good result, still some further improvement would be needed to fit our case.

As the model is 100% accurate in color recognition, we utilize this feature to add consistency to the classification of piece types. We implement a correction algorithm where the new recognized piece type is cross-checked with the last known correct board detection for consistency. As the model's color prediction is always correct, such a mechanism correctly adjusts for incorrect piece types by cross-verifying them with past detections.

This method of correction eliminates abnormal detections for the most part, providing safe detection performance. An average of 1.3 images must be processed per correct detection, and because each detection cycle requires one second, the ChessBot remains very responsive and efficient at all times.

IX. Room for Improvements

The current robotic chess coach is a functional prototype; however, certain features can be improved. One of the ways we can improve the system is by increasing its educational

features further. Hence, we are planning to include a chatbot in the dashboard which will interact with the users and provide them with more developed feedback on their moves. In addition, the Raspberry Pi can be shared among multiple arms as explained above, and four units can be attached to a single chip. This means that, using tools like FreeRTOS or POSIX, we can lower the cost of the robotic arms. There will be one Arduino per device to manage motors, and a single Raspberry Pi as a conductor for them. The communication will be through I2C or UART, and the optimization of task execution will be through a scheduling system. The system used will provide the synchronization of the arms and prevent processing bottlenecks. The scheduling system is time slot allocation for each arm, in which the commands are sent sequentially, so as not to overburden the Raspberry Pi. Furthermore, the old solar panel system allows us to recharge batteries when the robot is off. In order to enable autonomous continuous operation, two separate battery packs will be introduced, so that one powers the robot while the other charges. A power management system controls the switching process between the two batteries. Basically, the solar panel will charge the lowest voltage battery via a charge controller, and an LM5050-1 circuit will ensure that only the higher voltage battery supplies power to the load. Finally, a MOSFET detects a low battery voltage and shifts charging to it. The innovations will make the system more economical, efficient, and eco-friendly.

REFERENCES

[1] K. Sincharoenkul, N. Tongtep, and L. Boonlamp, "Supervised Classification of Board Games for Active Learning to Enhance Business Knowledge and Skills," 2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), Jun. 2020, doi: https://doi.org/10.1109/ecticon49241.2020.9158301.

[2] Ishika Khokhani, J. Nathani, Prem Dhawane, S. Madhani, and K. Saxena, "Unveling Chess Algorithms Using Reinforcement Learning and Traditional Chess Approaches in AI," pp. 1–4, Aug. 2023, doi: https://doi.org/10.1109/asiancon58793.2023.10269900.

[3] J. Madake, C. Deotale, G. Charde, and S. Bhatlawande, "CHESS AI: Machine learning and Minimax based Chess Engine," IEEE Xplore, Jan. 01, 2023. https://ieeexplore.ieee.org/abstract/document/10080746 (accessed May 03, 2023).

[4] Rajasekaran N, Raajkumar Sendilkumar, Kowshika Elangovan, and G. P. Arumugam, "Integration of Android Application and Smart Conventional Chess Board," vol. 7, pp. 561–566, Mar. 2023, doi: https://doi.org/10.1109/icsmdi57622.2023.00106.

[5] Andrew Tzer-Yeu Chen and Kevin I-Kai Wang, "Computer vision based chess playing capabilities for the Baxter humanoid robot," Apr. 2016, doi: https://doi.org/10.1109/iccar.2016.7486689.

[6] "(PDF) Gambit: An autonomous chess-playing robotic system," ResearchGate, doi: https://doi.org/10.1109/ICRA.2011.5980528.

[7] R Srivatsan, S Badrinath, and G. Lakshmi Sutha, "Autonomous chess-playing robotic arm using Raspberry PI," 2021 International Conference on System, Computation, Automation and Networking (ICSCAN), pp. 1–6, Jul. 2020, doi: https://doi.org/10.1109/icscan49426.2020.9262351.

[8] D. Lukac, "Playing chess with the assistance of an industrial robot," 2018 3rd International Conference on Control and Robotics Engineering (ICCRE), Apr. 2018, doi: https://doi.org/10.1109/iccre.2018.8376423.

[9] P. Karia, V. Jain, M. Shah, and S. Rane, "Digitization of Chess Board and Prediction of Next Move," 2022 IEEE 7th International conference for Convergence in Technology (I2CT), pp. 1–5, Apr. 2022, doi: https://doi.org/10.1109/i2ct54291.2022.9825379.

[10] A. Rossi, N. Ahmed, S. Salehin, T. H. Choudhury, and G. Sarowar, "Real-time Lane detection and Motion Planning in Raspberry Pi and Arduino for an Autonomous Vehicle Prototype," arXiv.org, 2020. https://arxiv.org/abs/2009.09391 (accessed Apr. 17, 2025).

[11] S. Björkqvist, "Estimating the Puzzlingness of Chess Puzzles," 2021 IEEE International Conference on Big Data (Big Data), pp. 8370–8376, Dec. 2024, doi: https://doi.org/10.1109/bigdata62323.2024.10825991.

[12] Andrew Tzer-Yeu Chen and Kevin I-Kai Wang, "Computer vision based chess playing capabilities for the Baxter humanoid robot," Apr. 2016, doi: https://doi.org/10.1109/iccar.2016.7486689.

[13] L. Vallance, "Exploring the Python Chess Module," 2018. Available: http://liamvallance.com/img/Exploring%20python%20chess.pdf

[14] S. Aggarwal and V. Ranga, "A Raspberry Pi 4 and Arduino-based Automated Intercom Door Lock System," 2024 1st International Conference on Advanced Computing and Emerging Technologies (ACET), pp. 1–6, Aug. 2024, doi: https://doi.org/10.1109/acet61898.2024.10730356.

[15] K. Pagonis, P. Zacharia, Antreas Kantaros, T. Ganetsos, and Konstantinos Brachos, "Design, Fabrication and Simulation of a 5-dof Robotic Arm using Machine Vision," vol. 113, pp. 1–4, Jun. 2023, doi: https://doi.org/10.1109/emes58375.2023.10171749.

[16] G. Franchini, S. Chiodini, M. Ghetti, and M. Pertile, "Mechatronic Design and Positioning Accuracy Characterisation of a Robotic Arm for Exploration Rovers," 2022 IEEE 9th International Workshop on Metrology for AeroSpace (MetroAeroSpace), pp. 452–457, Jun. 2023, doi: https://doi.org/10.1109/metroaerospace57412.2023.10190044.

[17] I. P. Okokpujie, A. O. Akinwumi, A. A. Adebayo, E. Omoiya, A. O. Adetunla, and K. Okokpujie, "Development of a Solar-Powered Robotic Arm With an Installed Solar Tracker," 2024 IEEE 5th International Conference on Electro-Computing Technologies for Humanity (NIGERCON), pp. 1–5, Nov. 2024, doi: https://doi.org/10.1109/nigercon62786.2024.10927046.

[18] A. Chaudhari, K. Rao, Krushna Rudrawar, Pratik. Randhavan, and P. Raut, "Development of Robotic Arm Prototype," Mar. 2023, doi: https://doi.org/10.1109/icscds56580.2023.10104928.

[19] Y. Xing, E. W. M. Ma, K. L. Tsui, and M. Pecht, "Battery management systems in electric and hybrid vehicles," Energies, vol. 4, no. 11, pp. 1840–1857, 2011, doi: https://doi.org/10.3390/en4111840.

[20] Y.-C. Chen, C.-Y. Hsieh, and C.-H. Lin, "A mutual blocking technology applied to dual power source switching control," Energies, vol. 12, no. 4, p. 576, 2019, doi: https://doi.org/10.3390/en12040576.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788, 2016, doi: https://doi.org/10.1109/cvpr.2016.91.

[22] B. Yang, J. S. Jin, F. Li, X. Han, W. Tong, and M. Wang, "A perspective correction method based on the bounding rectangle and least square fitting," 2021 18th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), pp. 260–264, Dec. 2016, doi: https://doi.org/10.1109/iccwamtip.2016.8079851.

[23] Y. Gui, Y. Wu, Y. Wang, and C. Yao, "Visual Image Processing of Humanoid Go Game Robot Based on OPENCV," Aug. 2020, doi: https://doi.org/10.1109/ccdc49329.2020.9164541.

[24] A. Nazir and Mohd. A. Wani, "You Only Look Once - Object Detection Models: A Review," IEEE Xplore, Mar. 01, 2023. https://ieeexplore.ieee.org/document/10112271 A. Nazir and M. A. Wani, "You Only Look Once - Object Detection Models: A Review,"

[25] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2019, doi: https://doi.org/10.1109/cvpr.2019.00075.

[26] T. T. Mac, T.-N. Dam, T. Tri, Ly Hoang Hiep, and T.-D. Nguyen, "The Development of an Intelligent Robot Arm Playing chess with Human-Robot Interaction," pp. 1–6, Dec. 2023, doi: https://doi.org/10.1109/ichst59286.2023.10565356.