Bangladesh University of Engineering and Technology

Department of Electrical and Electronic Engineering

Course No.: EEE 404

Course Title: Robotics and Automation Laboratory

Group No.: 02

Project Group No.: 01

Project Title: Chessbot: An Intelligent Robotic System for Playing Chess

Submitted by-

1. Shadid Yousuf (1906113)
2. Kazi Abrar Mahmud (1906112)
3. U Mong Sain Chak (1906195)
4. H. A. Hassan Tahmid (1806129)

Table of Contents

Abstract

A robotic manipulator is very often the most important part of a robotic system. It is customary to ensure precise movement control of the robotic manipulator, or simply put, robotic arm, to apply it to various real-life applications related to industrial automation, healthcare, home automation and so on. However, ensuring precision in control of a robotic arm is not an easy task, as it requires extensive knowledge about the measurements of links and joints, as well as necessary software support. In this project, we used a 5 DoF mechanical robotic arm and created its Unified Robotics Description Format (URDF) file from scratch. The URDF file led us to build a fully functional robotic arm capable of maintaining precise kinematic control. We demonstrated the arm's precision by deploying it to play chess physically. For that purpose, we developed an end-to-end pipeline that uses computer vision and deep learning integrated with Stockfish chess engine that predicts the most fruitful chess move followed by the player's move, and the robotic manipulator does the work of moving the pieces accordingly.

Introduction

A robotic arm is a mechanical device comprised of links, joints and manipulators designed to carry out laborious tasks, mitigating the need for human labor [1]. It is often the key part of robotic systems, performing a variety of tasks with precision, speed, and accuracy [2]. The flexibility of movement of the robotic arm depends on its degree of freedom (DoF). DoF refers to the number of independent movements a robotic arm can make and higher the DoF, higher the flexibility of the robotic arm [3]. However, in order to control the robotic arm for achieving precision in its operation, it is mandatory to correctly take the kinematics associated with it, the relationship between its joint alignment and spatial layout, into account [4]. Higher degree of freedom has this tradeoff between higher flexibility and more complicated kinematics [5]. Two kinematics calculation are required for precision in control of the robotic arm, namely, forward kinematics and inverse kinematics. For a robotic arm, forward kinematics involves using joint angles as input to compute the Cartesian position and orientation of the end-effector, which is the tool or device attached to the arm's end, while inverse kinematics for a robotic arm determines the joint angles required to achieve a given Cartesian position and orientation of the end-effector [6].

Although calculations involving forward kinematics is straightforward, inverse-kinematics calculation is a challenging task [7]. There are two commonly used approaches for solving inverse kinematics equation: analytically, which is precise but complex for large joint chains, and iteratively, which simplifies and evaluates the equation repeatedly, using the Jacobian matrix for accurate approximations [8].

Nowadays many software and packages provide assistance in the meticulous calculation of kinematics by parsing the Unified Robotics Description Format (URDF) file. It is a format used to describe the physical configuration of a robot in a standardized way. It includes the robot's geometry, kinematics, and dynamics, which are essential for simulating and controlling robots in software frameworks [9]

A robotic arm capable of precise control deployed in recreational activities such as playing chess is not a new concept [10][11][12]. However, the previous attempts required either an extensive hardware setup often unavailable or too costly to install from Bangladesh perspective, or advanced software frameworks often foreign to amateur developers in Bangladesh. In contrast to that, we developed a low-cost autonomous chess-playing robot with easily available hardware and familiar software frameworks. Our primary contributions in this project are:

1. Creating an URDF file from CAD design for a commonly used 5-DoF robotic manipulator
2. Introducing an alternative approach for precision in control using Python
3. Establishing a simple deep-learning-aided computer vision pipeline for playing chess
4. Confining the cost of the project within affordability of amateur robotics enthusiasts (12,300 BDT, 100 USD)

Design

The initial objective for applying kinematics to the robotic manipulator is creating the URDF file for the specific robotic arm being used. That is why, the whole arm is designed in Autodesk Fusion 360 software and saved as a (.f3d) file. The design is shown in Fig 1.

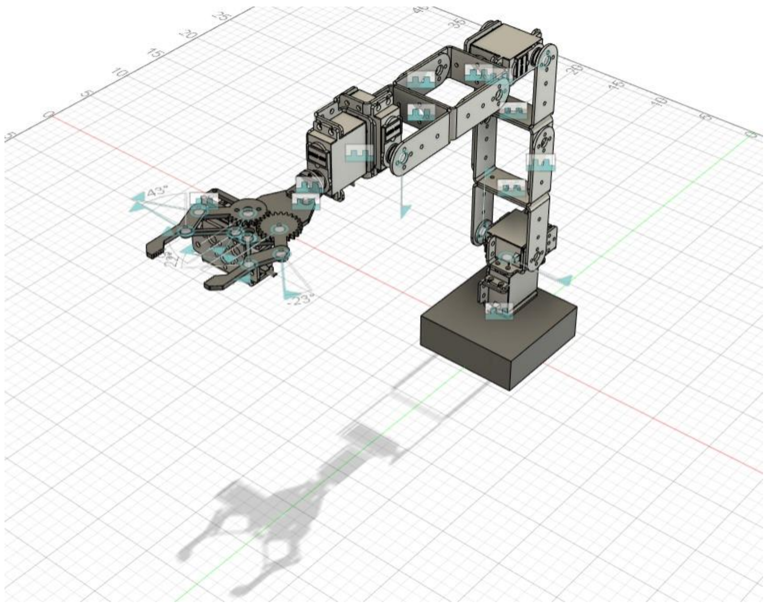


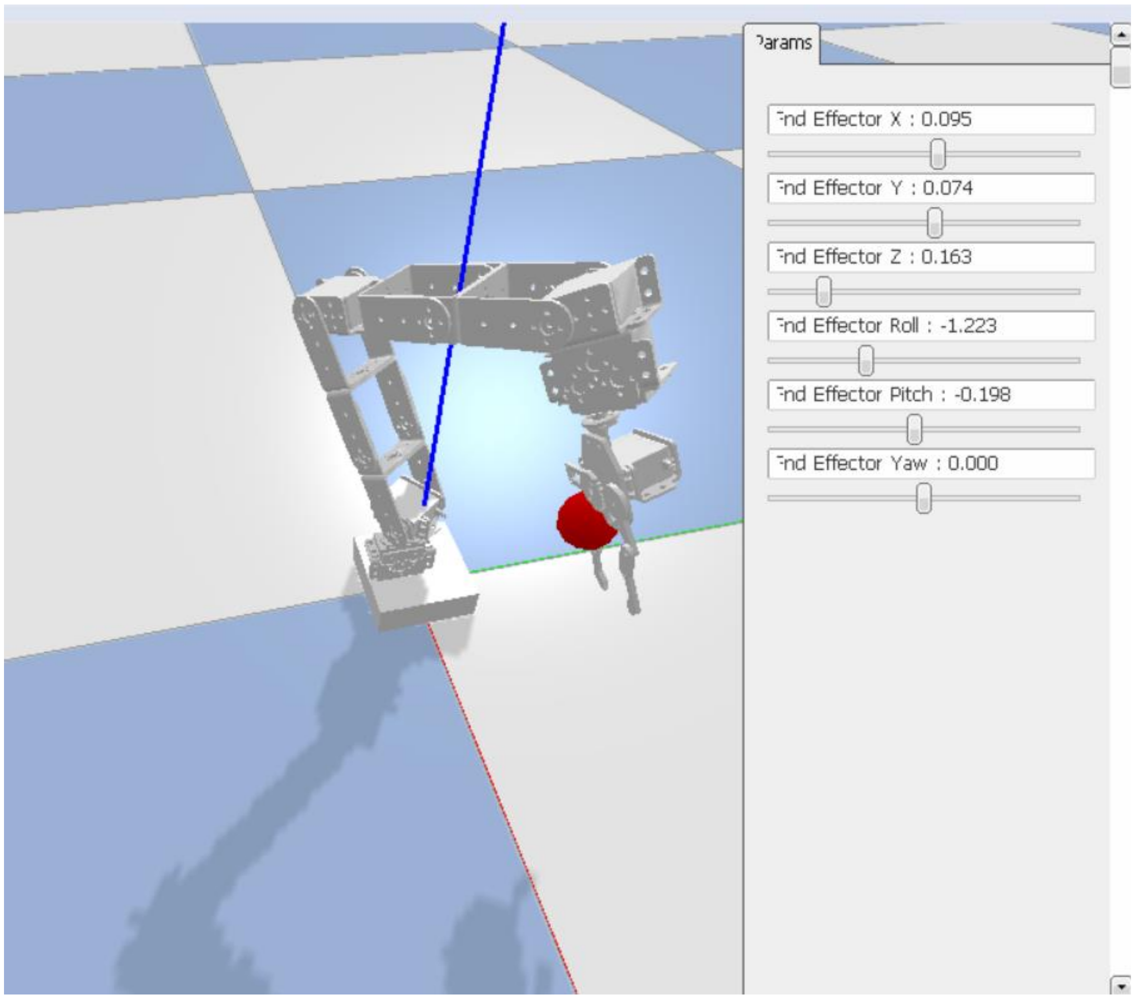
Fig. 1: CAD design of the robotic manipulator in Fusion 360

The (.f3d) file is then used to generate the URDF file.

It should be noted that we added two extra links for expanding the reach of our arm, so that it is able to reach to the furthest grids of the chessboard.

Simulation

The URDF file allows to apply forward and inverse kinematics and simulate the results. All the operations are carried out in Python platform. Our choice of simulation platform was the PyBullet. PyBullet is a fast and easy to use Python module for robotics simulation and machine learning, with a focus on sim-to-real transfer. With PyBullet it is possible to load articulated bodies from URDF file format, apply forward and inverse kinematics and simulate in real time. Fig 2 shows the Graphical User Interface(GUI) for the simulation of our robotic manipulator in PyBullet.



End Effector X : 0.095
End Effector Y : 0.074
End Effector Z : 0.163
End Effector Roll : -1.223
End Effector Pitch : -0.198
End Effector Yaw : 0.000

Fig. 2: Simulation of the robotic manipulator in PyBullet GUI.

In the GUI, the X, Y and Z sliders can be used to set the target position of the end effector in Robot Coordinate System in meters. The red sphere represents the target position. The yaw, roll and pitch sliders can be used to set the orientation of the end effector in radians, which then is translated to quaternion representation for solving the inverse kinematics with the preset orientation. In our case, it was suitable to set the pitch at -1.57 radians (-90 degrees), while roll and yaw were set to zero.

Inverse Kinematic Solution Method: Jacobian Iteration

So far the most popular way solving the inverse kinematics is the Jacobian Iteration method. The Jacobian Iteration Method for Inverse Kinematics (IK) is a widely used numerical approach to solve for joint configurations of a robotic manipulator to achieve a desired end-effector position and orientation. This method relies on the Jacobian matrix J, which relates the joint velocities $\dot{\theta}$ to the end-effector velocities $\dot{x}$ using the linearized relationship:

$$\Delta\dot{x} \approx J(\theta) \cdot \Delta\theta$$

The error vector ("e"), representing the difference between the desired and current end-effector pose, is iteratively minimized. The joint variable update is computed as:

$$\Delta\theta = J^{+}(\theta) \cdot e$$

where $J^{+}(\theta)$ denotes the pseudo-inverse of the Jacobian. Subsequently, the joint configuration is updated as:

$$\theta\_new = \theta\_current + \Delta\theta$$

To handle singularities or near-singular configurations, techniques such as Damped Least Squares (DLS) are employed, modifying the pseudo-inverse computation to:

$$\Delta\theta = (J^{T} J + \lambda^{2} I)^{-1} J^{T} \cdot e$$

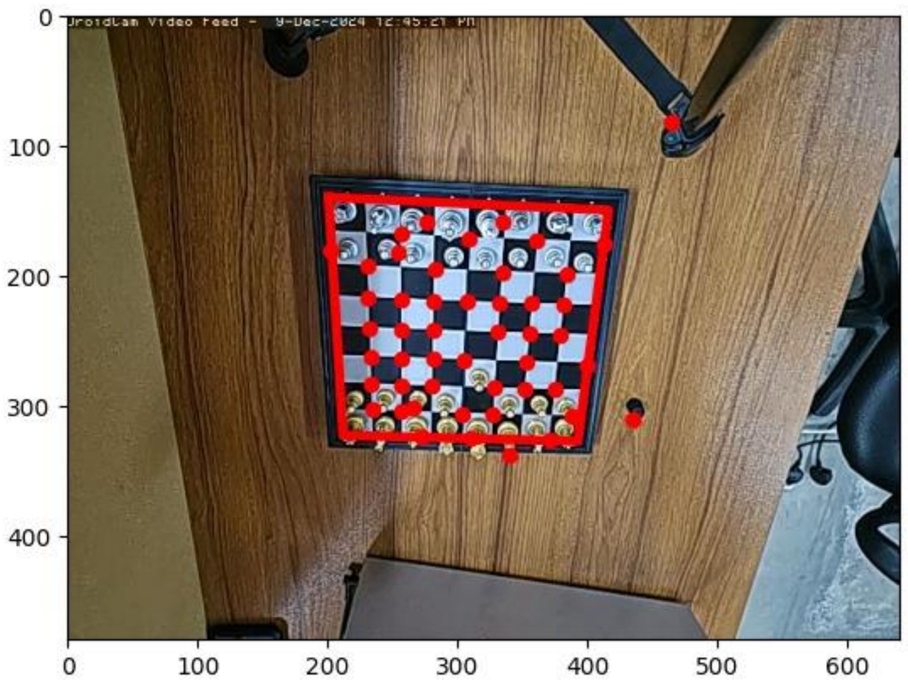
where λ is the damping factor. The process iterates until the error norm ($\|e\|$) is below a predefined threshold. This approach, though sensitive to singularities and nonlinearities, remains a cornerstone technique for robotic manipulators in real-time control and path planning.

Obtaining Chessboard State and Next Move Prediction

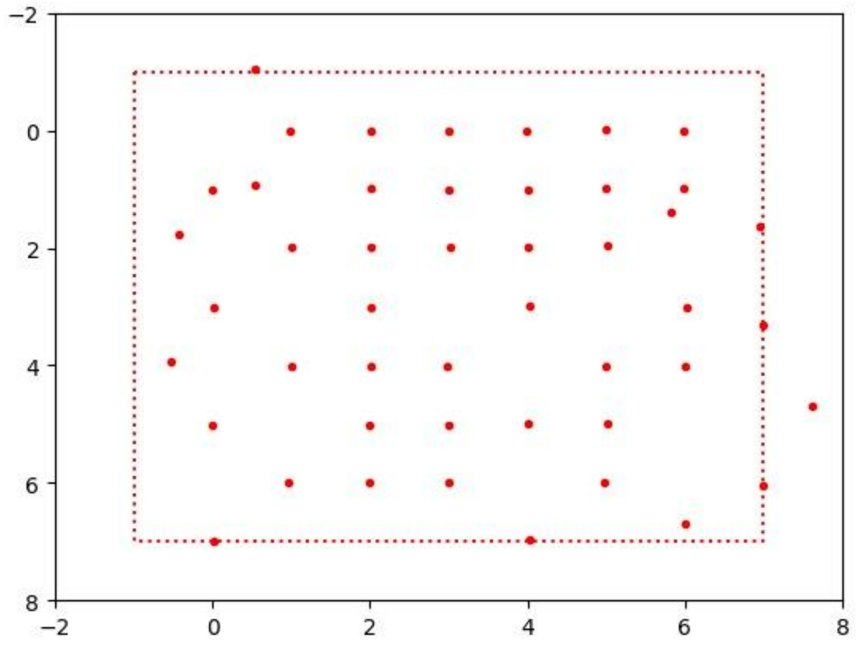
At first, the camera feed was transferred from the mounted camera to the Python platform using DroidCam. From the raw board image, saddle points were detected and filtered out, followed by homographic transformation of the image. After that, the transformed image was divided into 64 patches, that is, each chessboard cell is represented by distinct patches. The patches are classified into 3 classes, namely, occupied by a white piece, occupied by a black piece or unoccupied, by a pretrained neural network model. After the patches are correctly classified, the move by the user can be determined by comparing the difference between the current frame and the previous frame. This information is fed to the

Stockfish Chess Engine, which predicts the next move to give by the robot. The obtained outcome of each of the described steps above is shown in Fig. 3.

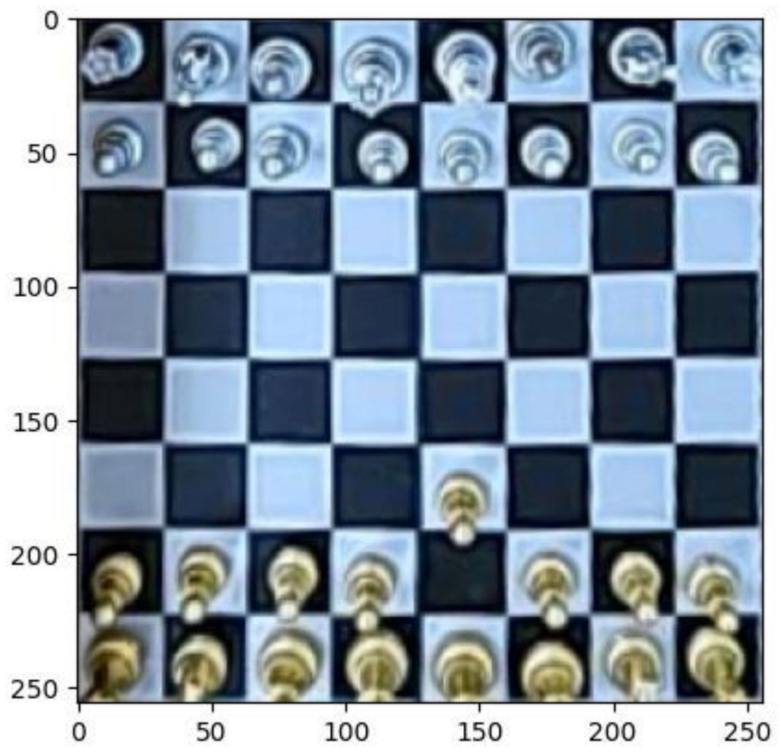
(a) Captured Chessboard



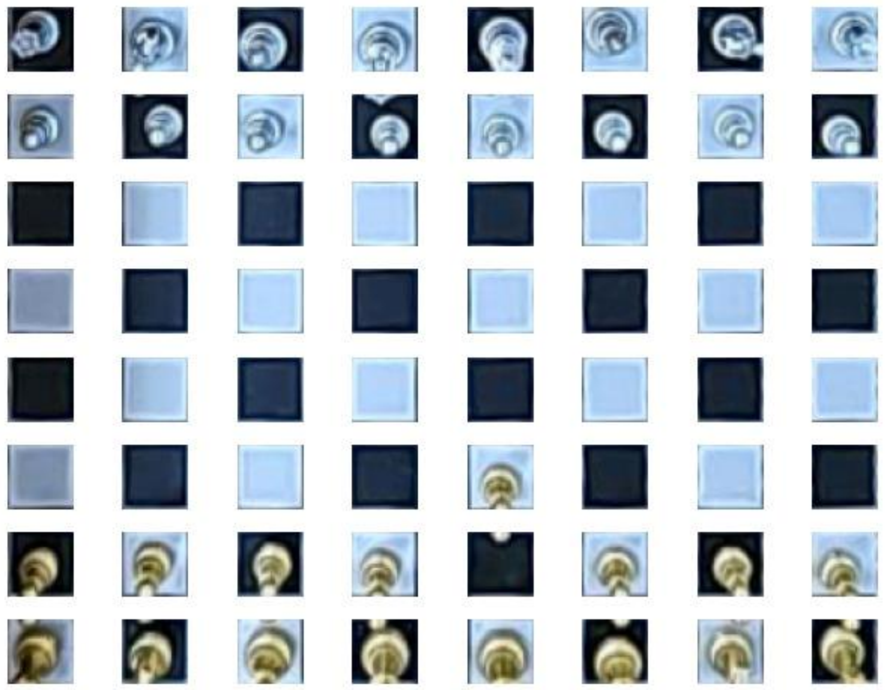
(b) Identifying Saddle Points



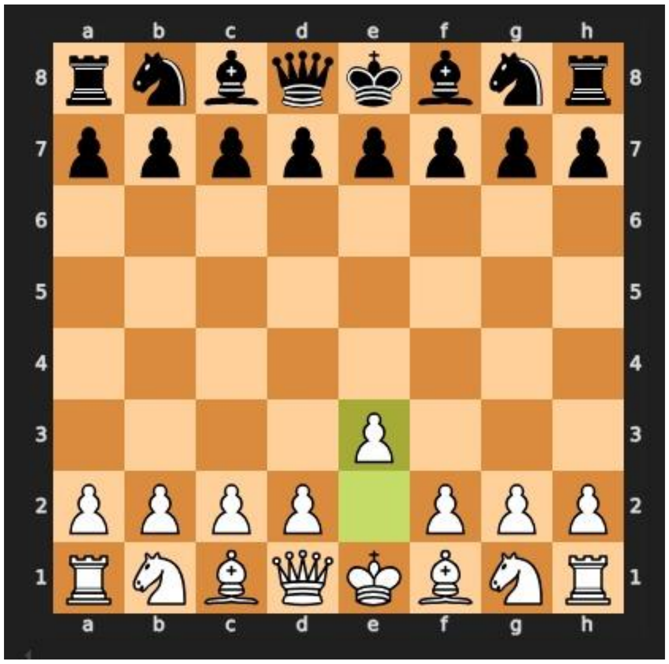
(c) Filtering Saddle Points



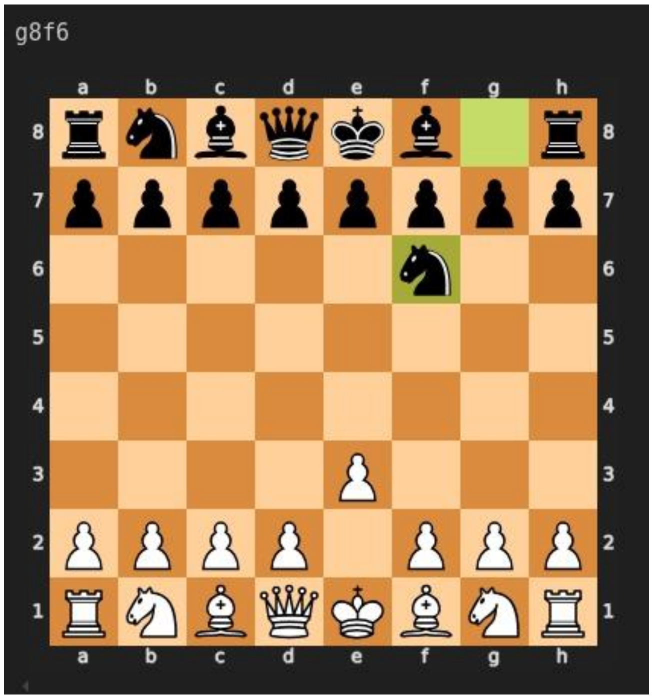
(d) Homographic Transformation



(e) Patch division



(f) Human move identification and Stockfish visualization



(g) Next move suggestion by Stockfish

Fig. 3: Pipeline for predicting next move from camera feed

Convolutional Neural Network (CNN) Architecture for Patch Recognition

Each of the patches can either be unoccupied, occupied by a white piece or occupied by a black piece. For correctly identifying that, we developed a dataset, labeled them and trained a simple neural network model. Fig. 4 shows the model architecture.

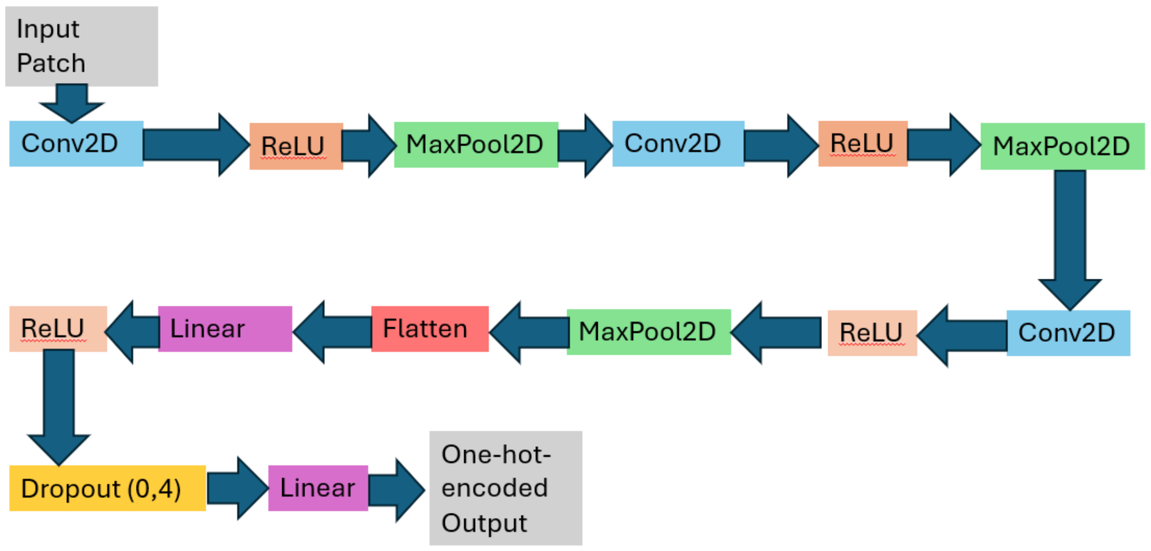


Fig. 4: Architecture of the Convolutional Neural Network (CNN) used for patch recognition

Combined Pipeline

Integrating all the described steps above, we get the overall pipeline of out chess playing robot. The overall pipeline is depicted in Fig. 5.

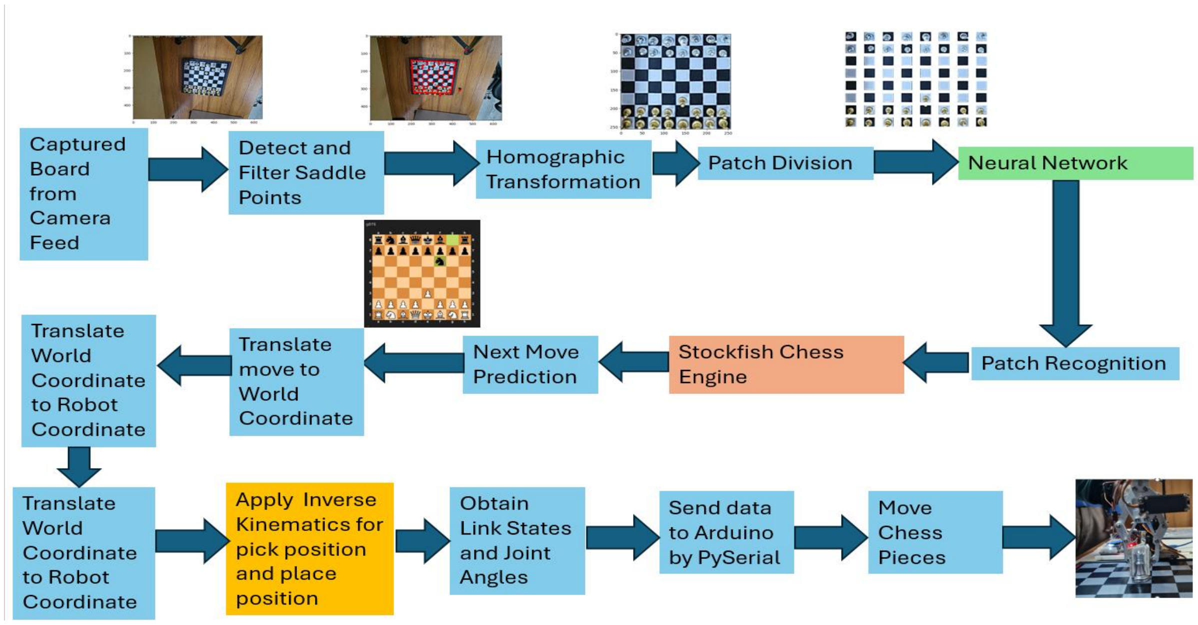

Captured Board from Camera Feed
Detect and Filter Saddle Points
Homographic Transformation
Patch Division
Neural Network
Patch Recognition
Stockfish Chess Engine
Next Move Prediction
Translate move to World Coordinate
Translate World Coordinate to Robot Coordinate
Translate World Coordinate to Robot Coordinate
Apply Inverse Kinematics for pick position and place position
Obtain Link States and Joint Angles
Send data to Arduino by PySerial
Move Chess Pieces

Fig. 5: Overall pipeline of the ChessBot

Practical Implementation

After assembling the robotic arm, it was wired with an Arduino Mega using the PCA9865 Servo Driver module. An LCD Display was added for observing the joint angle values obtained from inverse kinematics calculation. For getting the state of the chessboard, a smartphone was mounted on a tripod to capture the chessboard. Each frame was transferred to the Python platform using DroidCam. The overall hardware setup of the project is shown in Fig. 6.

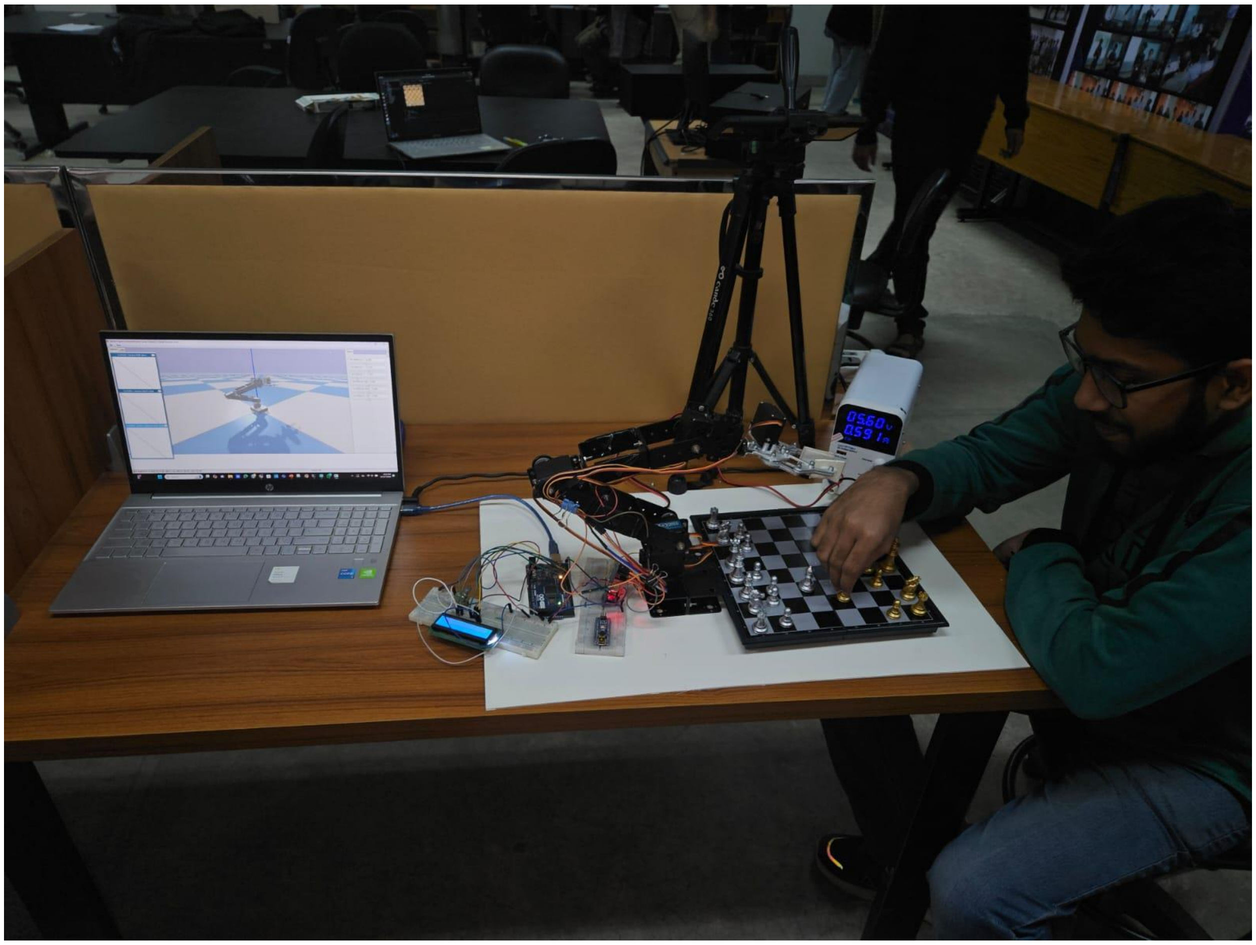


Fig. 6: A person playing chess with the ChessBot

Hyperparameter Tuning

Through experimentation, several hyperparameters are tuned to achieve the optimal performance.

The servo motors operate by a PWM signal, the angle of the servo motor is determined by the duty cycle of the PWM signal. The frequency of the PWM signal was set to 50 Hz. In the robotic manipulator, two different types of servo motors were used. For the second joint (neck), a high-torque DS3235 Digital Servo Motor was used, and for the rest of the joints MG996R motors were used. For the former one, the minimum and maximum pulse count out of 4096 was set to 170 and 530, respectively. For the rest of the motors, the range is from 120 to 520.

Each servo motor requires a running current of 500mA, so a power supply unit is required that can continuously supply a current of 500 mA X 6 = 3A. The power supply voltage was kept at 5.5V.

For optimum grabbing, the pitch of the end effector was set at -90 degrees, while the yaw and roll were kept at zero. The height or z-coordinate was kept at 10cm, to avoid tumbling of other pieces while aiming for a particular piece.

The chessboard we used had a grid-spacing of 2.8cm. The board was kept exactly 7.3cm(y-offset) from the center of the base of the arm. These values are needed to determine the world coordinate positions and the robot coordinate locations for pick-and-place operation. The x and z coordinate values of world and robot coordinate systems were the same; one simply needs to add 7.3 cm to the y-value of world coordinate to transform to the robot coordinate system.

The margin of error for inverse kinematic calculation was set to 0.1 cm. That means the Jacobian iteration was carried out until the end effector position was 0.1cm within the target position.

Challenges Faced and How We Overcame Them

1. **Unable to Stay Upright:** Due to our modification in the link length, the robotic arm was unable to being pulled down by the heavy weight for positions for which the center of gravity of the arm was needed to be much further from the base. To address this issue, we replaced the servo motor in the second joint (neck) with a high-torque (35 kg/cm) alternative, which solved our problem.

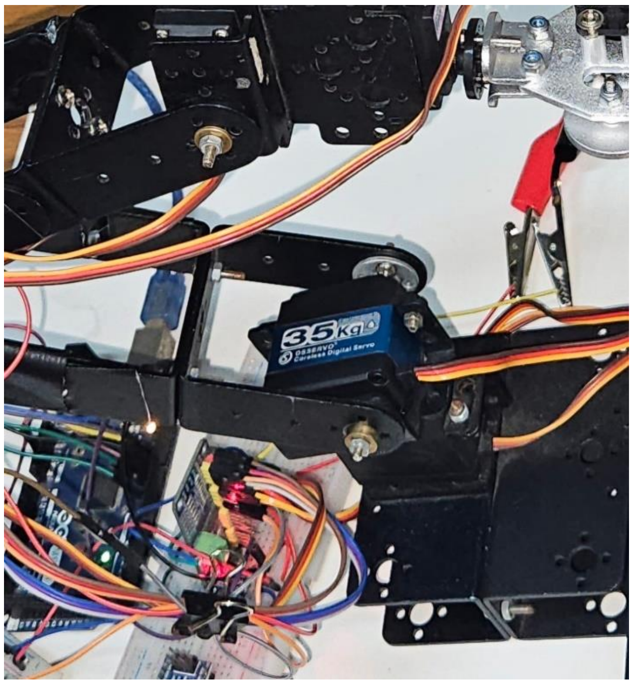
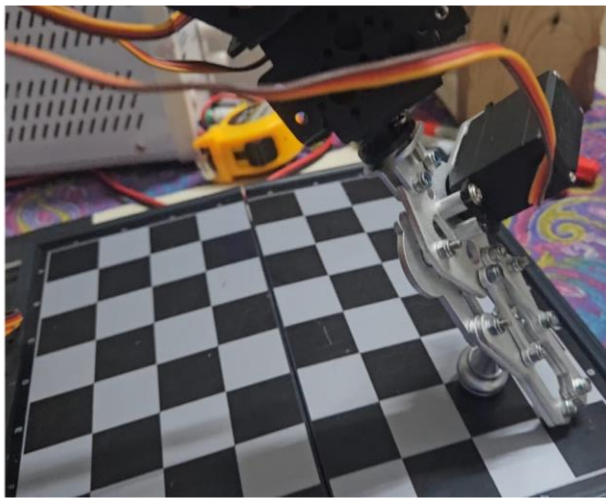


Fig. 7: A high-torque servo motor was used in the neck joint

1. **Inclined grasping by the end effector:** Our initial inverse kinematics solution would require the end effector to grab any chest piece with an inclined orientation, as shown in Fig. 7(a). This would tumble all the nearby chess pieces, so we set the pitch value to -90 degree afterwards for vertical picking. However, this operation was also unsuccessful most of the cases because the tip of the end effector has a low surface area, often unable have a firm grip on the chess piece. We then cut out two small chips cut out from plastic wood, and glued rubber slices on the outer surface for ease of grasping. Fig. 8(b) shows the modified end effector with vertical grasping operation.


(a) Picking with an inclined end-effector orientation

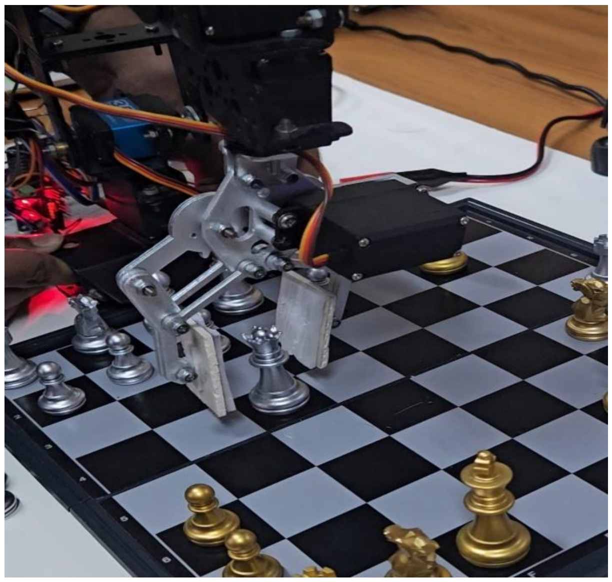

(b) Vertical picking after modification

Fig. 8: Modification of the end effector

2. **Low-resolution Webcam Feed:** At first for the computer vision part, we used a USB webcam. However, the algorithm failed to detect the saddle points properly as the captured image from the webcam was of low resolution. Instead of using a higher resolution webcam, we decided to utilize a smartphone camera as the image resolution is typically higher in smartphones. The captured image was wirelessly transmitted over WiFi network to the python script by a software named DroidCam.
3. **Misdetected Cells:** Our first attempt at getting the chessboard state involved getting the difference of the previous frame from the current frame after

homographic transformation. In this method, white pieces occupied in white cells were often misdetected as empty cells. This led us to incorporate patch division and the use of a neural network model for proper identification of occupied cells, as already discussed.

Room for Improvement

Path Planning: In our project the robotic arm uses solutions from uniformly spaced Jacobian iterations while the solution gradually converges into the target. Though this ensures enough subtlety for the operations of this project, the trajectory of the end effector can be more polished by incorporating advanced path-planning algorithms such as Probabilistic Roadmap (PRM), Rapidly Exploring Random Tree (RRT) etc.

Closed-loop Control: By incorporating closed-loop feedback control, more precision in arm operation can be achieved. This might require motors with encoder for positional feedback, integrated with an Inertial Mass Unit (IMU), or the incorporation of Reinforcement Learning.

Deep Grasp using Reinforcement Learning: In our project, the end effector grasps the target object with a fixed end-effector orientation. This might not be suitable for target objects for tricky positions. For that reason, reinforcement techniques can be applied to achieve deep grasping: grasping the target object not with a fixed orientation but with an optimized orientation that varies with respect to target location and orientation.

Unique Aspects of Our Project/ Novelty

Reproducibility:

From the planning phase of this project, our focus was on ensuring reproducibility, particularly for robotics enthusiasts in Bangladesh. The robotic arm, widely used in various robotics projects in the country, lacked a publicly available and accurate URDF file. To address this, we created a corrected URDF file based on the arm's CAD design and made it accessible to the public. This resource enables users to achieve precise control of this robotic arm in their own projects and solutions.

In general, for precise kinematic control of a robotic system, Robot Operating System (ROS) offers built-in packages and flexibilities to ensure smooth operation. However, ROS is only supported in specific versions of Ubuntu operating system (OS). Most people in Bangladesh has Windows installed and is not accustomed to Ubuntu and hence, ROS. This is a great barrier towards developing high-level projects. To address this issue, we bypassed ROS and simply used Python scripts instead, and showed a way of achieving high-level precision in inverse-kinematics operation.

All other hardware components are easily available in Bangladesh.

Scalability:

Needless to say, although the robotic manipulator is used to play chess in this project, it can be utilized in all sorts of robotic applications such as trash collector robot, pick-and-place robot, object sorter robot etc., to name a few.

Cost Efficiency:

The components were chosen in such a way that the cost remained low. For example, the task would be much easier to use intelligent development board such as Raspberry Pi that can run Python scripts directly. Raspberry Pi is often too costly for an amateur robotics enthusiast. That is why we decided to use Arduino, a much cheaper option. All the necessary calculations were done within the Python platform, and the data was then sent to Arduino by establishing serial communication between Arduino and Python. Overall the cost of the total project was brought down to around 12000 Tk (100 US Dollars).

Societal Benefits

1. **Allows the participation of handicapped people in chess tournaments:** A handicapped or injured participant is not able to play chess in the competitive tournaments. Though our robot is programmed to play against a human and the move is predicted by a chess engine, it can be easily reprogrammed so that it listens to the move to give by a verbal command and make the move on behalf of the physically impaired player. This way such participants can enjoy the game of

chess anywhere, be it at home physically playing with someone or at a chess tournament.

2. **Can be reprogrammed to provide mobility support to the disabled/injured people:** A robotic arm with precise control has countless area of applications. Our ready-arm can be easily reprogrammed to be controlled using voice command, eye-tracking, sign-language or Brain-Computer Interface (BCI), just to name a few. Therefore, it has the scope of providing mobility to the injured or handicapped personnel to make their daily life a little bit better.

GitHub Repository

The URDF file, along with all the codes used in this project can be found in the following GitHub Repository:

https://github.com/ShadidYousuf/ChessBot-ChessBot-An-Intelligent-Robotic-Companion-For-Playing-Chess

List of References

[1] Gracie, "The Role of Robotic Arms in Modern Manufacturing," JHFOSTER, Sep. 19, 2024. https://jhfoster.com/automation-blogs/the-role-of-robotic-arms-in- modern-manufacturing/

[2] "Industrial Robotic Arm Overview," Intel, Mar. 2020. https:/www.intel.com/content/www/us/en/robotics/robotic-arm.html

[3] "What are degrees of freedom in robotic arms? (Easy guide) - Standard Bots," Standardbots.com, 2024. https://standardbots.com/blog/degrees-of-freedom

[4] "Kinematics," motion.cs.illinois.edu. https://motion.cs.illinois.edu/RoboticSystems/Kinematics.html

[5] S. Pan and G. Endo, "Toward mission-dependent long robotic arm enhancement: design method of flying watch attachment allocation based on thrust drivability," ROBOMECH Journal, vol. 8, no. 1, Mar. 2021, doi: https://doi.org/10.1186/s40648-021-00198-1.

[6] L. Joseph, "Robot Kinematics in a Nutshell," ROBOCADEMY, Apr. 20, 2020. https://robocademy.com/2020/04/21/robot-kinematics-in-a-nutshell/

[7] D. L. Piper, “The Kinematics of Manipulators Under Computer Control,” Thesis, Computer Science Department, School of Humanities and Sciences, Stanford University, 1968.

[8] R. Nilsson, “Inverse kinematics,” DIVA, 2016, doi: https://doi.org/1018821/FULLTEXT01.

[9] “Introduction to URDF — Industrial Training documentation,” industrial-training-master.readthedocs.io. https://industrial-training-master.readthedocs.io/en/melodic/_source/session3/Intro-to-URDF.html

[10] F. Abdullah, T. Al-Saedi, and A. Mohammed, “Design and Implementation of Chess-Playing Robotic System.” Available: https://ijcset.net/docs/Volumes/volume5issue5/ijcset2015050501.pdf

[11] Musa Atas, Y. Dogan, and Ssa Atas, “Chess playing robotic arm,” Apr. 2014, doi: https://doi.org/10.1109/siu.2014.6830443.

[12] S. Sarker, “Wizard chess: An autonomous chess playing robot,” Dec. 2015, doi: https://doi.org/10.1109/wiecon-ece.2015.7443971.

APPENDIX A

Hardware Requirements and Cost Breakdown

Name of the Component	**Quantity**	**Unit Cost (BDT)**	**Cost (BDT)**
Arduino Mega	1	1700	1700
5 DoF Mechanical Robotic Arm Kit (Aluminium)	1	3500	3500
MG996R Servo Motor (Torque: 11 kg/cm)	5	300	1500
DS3235 Digital Servo Motor(Torque: 35 kg/cm)	1	3000	3000
PCA 9685 Servo Motor Driver	1	500	500
Magnetic Chessboard	1	600	600
Power Supply Unit	1	1300	1300
Miscellaneous			200
			Total 12300 BDT

APPENDIX B

Software Dependencies and Necessary Modules

1. Autodesk Fusion 360
2. Python (3.8.0 or higher version preferrable)
3. Stockfish Chess Engine
4. PyBullet Physics Module
5. DroidCam
6. Arduino IDE
7. PySerial

APPENDIX C

Tuned Hyperparameters for Optimal Performance

Name of the Hyperparameter	Set Value
End effector pitch	-90n degrees
End effector yaw	0 degree
End effector roll	0 degree
End effector grasping height	10 cm
Range of Pulse Length Count (out of 4096): MG996R	[120,520]
Range of Pulse Length Count (out of 4096): DS3235	[170,530]
PWM Frequency	50 Hz
Baud Rate for Serial Communication	9600 bps
Data transmission interval	0.2 seconds