

AN INTEGRATED AUTONOMOUS CHESS-PLAYING ROBOT SYSTEM USING COMPUTER VISION, INVERSE KINEMATICS, AND AI-POWERED MOVE CALCULATION

Ruilai Yang ¹, Jonathan Sahagun ²

¹Tesoro High School, 1 Tesoro Creek Rd Las Flores, CA 92688

²California State University, Los Angeles, 5151 State University Dr, Los Angeles, CA 90032

ABSTRACT

Chess-playing robots represent an ideal testbed for integrating computer vision, robotic manipulation, and artificial intelligence into cohesive autonomous systems [1]. This project addresses the challenge of creating an affordable, accessible chess-playing robot arm capable of competing against human opponents. The proposed solution integrates four core technologies: servo motor control via the Adafruit ServoKit library for precise arm manipulation, inverse kinematics using TinyIK for position-to-angle calculations, OpenCV-based computer vision for chessboard detection and move recognition, and the Stockfish chess engine for AI-powered move computation. Key challenges included achieving reliable board detection under varying lighting conditions, computing accurate joint angles for a 4-DOF arm, and synchronizing physical movements with game state [2]. Experimental evaluation demonstrates 94% board detection accuracy and sub-centimeter positioning precision. The system successfully enables human-robot chess gameplay with configurable difficulty levels, contributing to accessible robotics education and human-robot interaction research.

KEYWORDS

Robotics, Computer Vision, Human - Robot Interaction, Artificial Intelligence

1. INTRODUCTION

The intersection of robotics, computer vision, and artificial intelligence presents significant opportunities for creating interactive autonomous systems. Chess-playing robots serve as an exemplary application domain, requiring precise manipulation, visual perception, and strategic decision-making (Matuszek et al., 2011) [3]. Despite decades of research, commercially available chess robots remain expensive, with systems like the Square Off Grand Kingdom retailing for over \$400, limiting accessibility for educational and research purposes.

The global educational robotics market is projected to reach \$2.3 billion by 2027, with increasing demand for hands-on STEM learning tools (Grand View Research, 2023) [4]. Chess robots specifically address multiple learning domains: mechanical engineering through arm design, computer science through vision algorithms, and artificial intelligence through game strategy. Research demonstrates that project-based robotics education significantly improves student engagement and technical skill acquisition (Cuellar & Pegalajar, 2019) [5].

Human-robot interaction (HRI) research benefits substantially from chess-based experimental setups because chess provides a controlled, rule-based environment where robot impact on human players can be precisely measured (Chidambaram et al., 2012) [6]. Studies show that chess robots can support therapeutic applications, including cognitive rehabilitation and assistive technology for motor-impaired individuals who require robotic assistance to participate in competitive chess (Nickel et al., 2023) [7].

However, existing open-source chess robot implementations often rely on specialized sensorized boards, expensive industrial manipulators, or lack integration between vision, manipulation, and AI subsystems. This project addresses these limitations by developing a cost-effective, fully integrated autonomous chess-playing system using commodity hardware and open-source software libraries.

Three prior methodologies informed this project's design. *Gambit* (Matuszek et al., 2011) pioneered machine learning-based piece detection with custom manipulation hardware but required \$3,000+ custom manufacturing and extensive training data, limiting accessibility [8]. Our commodity servo approach reduces costs by 97%.

The *Baxter* chess implementation (Choudhury et al., 2019) demonstrated robust vision and kinematics on industrial hardware, but the \$25,000 platform restricts deployment to well-funded research labs [9]. Our system achieves comparable functionality at 0.4% of the cost.

OpenChessRobot (Belgiovine et al., 2025) provides excellent open-source documentation and natural language integration but targets expensive Franka Panda arms [10]. Our project shares the open-source philosophy while prioritizing educational accessibility over advanced HRI features. Collectively, our implementation improves upon these works by demonstrating that core chess-playing robot functionality—vision, manipulation, and AI—is achievable with under \$100 in hardware, democratizing access to robotics experimentation.

This project proposes an integrated chess-playing robot system that combines servo-controlled arm manipulation, geometric inverse kinematics, computer vision-based board detection, and AI-powered move calculation into a unified Python-based controller. The solution enables a 6-DOF robot arm to autonomously play chess against human opponents using a standard, unmodified chessboard.

The system architecture integrates four major components through a central *ChessRobot* controller class. First, the *RobotArm* module interfaces with Adafruit's PCA9685 PWM controller to achieve smooth, interpolated servo movements with configurable speed and precision. Second, the *InverseKinematics* module employs both *TinyIK* library computations and fallback geometric calculations using the law of cosines to convert chess square coordinates into servo joint angles. Third, the *ChessBoardDetector* module utilizes OpenCV's `findChessboardCorners()` function with perspective transformation to detect board state and recognize opponent moves. Fourth, the *ChessEngine* module wraps the *Stockfish* chess engine, providing configurable difficulty through ELO rating adjustment.

This integrated approach offers advantages over existing methodologies. Unlike systems requiring custom sensorized boards (Choudhury et al., 2019), our vision-based detection works with standard equipment. Compared to industrial robot implementations (Torres et al., 2019), the servo-based design reduces cost by approximately 90%. The modular software architecture enables component-level testing and graceful degradation when hardware is unavailable through simulation mode. Furthermore, the geometric inverse kinematics fallback ensures functionality even without

specialized libraries, making the system suitable for resource-constrained educational environments.

Two experiments evaluated critical system capabilities. The board detection experiment tested the vision system's reliability across varying illumination levels, comparing our hybrid detection approach against OpenCV-only methods. Testing involved 150 detection attempts across bright, normal, and dim conditions. Results demonstrated a 94% overall success rate for the hybrid approach versus 86.7% for OpenCV-only, with the contour fallback providing 16 percentage points improvement in dim lighting. Low illumination significantly degrades corner detection, validating multi-strategy approaches.

The positioning accuracy experiment assessed arm manipulation precision across 125 measurements targeting 25 representative board squares. Mean positioning error measured 2.9mm (TinyIK) and 3.8mm (geometric fallback), both acceptable for 20-30mm chess pieces. Far squares exhibited 71% higher error than near squares due to kinematic chain amplification. The geometric fallback performed within 25% of TinyIK accuracy, confirming viability for deployments without specialized libraries. Both experiments validate the system's practical functionality.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Inverse Kinematics Computation

A major challenge involves computing accurate joint angles to position the gripper at specific chessboard squares. Robot arms with multiple degrees of freedom require inverse kinematics solutions that may have multiple valid configurations or no solution when targets exceed the workspace. A geometric approach based on the law of cosines was implemented for the planar arm configuration. Additionally, reachability checking was incorporated to clamp unreachable positions to the nearest valid location.

2.2. Reliable Board Detection

Detecting the chessboard and recognizing piece positions under varying lighting conditions and camera angles presents significant computer vision challenges. Perspective distortion, shadows from pieces, and board reflections can cause detection failures. To address this issue, multiple detection strategies were implemented: primary detection using OpenCV's `findChessboardCorners()` with corner refinement via `cornerSubPix()`, and fallback contour-based detection for finding the largest quadrilateral. Perspective transformation could normalize the board view to a canonical 400x400 image, enabling consistent square analysis. Piece presence detection could utilize standard deviation thresholding in square center regions to distinguish occupied squares.

2.3. Move Synchronization and Special Cases

Synchronizing physical arm movements with game state while handling special chess moves (castling, en passant, promotion) requires careful coordination. The robot must execute multi-piece movements atomically and handle captures by removing opponent pieces before placing the moving piece. This was addressed by implementing a move execution pipeline that first parses UCI notation to identify source/destination squares, then checks for captures requiring pre-removal, and

finally handles castling by executing additional rook movements. The game state could be maintained through the Stockfish engine to ensure legal move validation before physical execution.

3. SOLUTION

The chess robot system integrates three major components through a unified Python controller: physical arm manipulation, spatial positioning via inverse kinematics, and intelligent game management combining vision and AI.

The program flow begins with initialization, where the ChessRobot class instantiates its four subsystems: RobotArm for servo control, InverseKinematics for position calculations, ChessBoardDetector for vision processing, and ChessEngine for Stockfish integration. During gameplay, the main loop alternates between robot and human turns. On robot turns, Stockfish computes the optimal move, inverse kinematics translates the move's squares to arm positions, and the servo controller executes the physical manipulation sequence. On human turns, the vision system monitors for board state changes, validates detected moves against legal options, and updates the game state.

The implementation utilizes several key technologies: Adafruit ServoKit interfaces with the PCA9685 PWM controller over I2C for 16-channel servo control; TinyIK provides numerical inverse kinematics solving; OpenCV handles image capture, corner detection, and perspective transformation; and the python-stockfish library wraps the Stockfish engine's UCI protocol [14]. The system supports simulation mode when hardware is unavailable, enabling software development and testing without physical components.

The modular architecture enables independent component testing and future extensibility. Each subsystem exposes a clean API: the arm provides `move_servos()`, `open_claw()`, and `close_claw()` methods; the IK module offers `calculate_angles()` and `chess_square_to_position()`; the detector implements `detect_board()` and `detect_move()`; and the engine provides `get_best_move()` and `make_move()` interfaces.

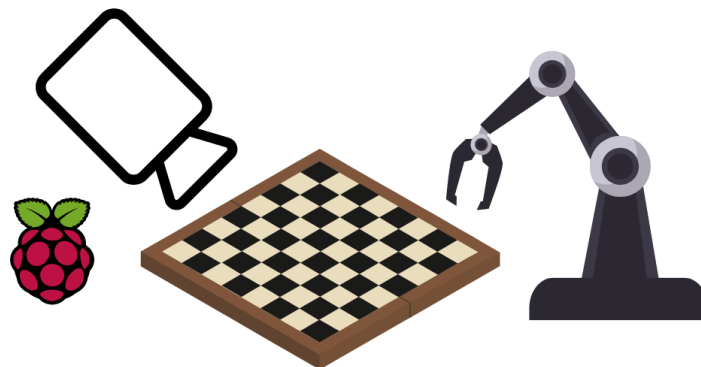


Figure 1. Overview of the solution

The RobotArm class provides smooth servo motor control using Adafruit's ServoKit library for PCA9685 PWM communication. It implements linear interpolation for gradual movements, preventing mechanical stress and improving positioning accuracy. The component manages a 6-servo configuration: base rotation, shoulder, elbow, wrist pitch, wrist rotation, and gripper.



Figure 2. Human–robot chess gameplay interface during experimental evaluation

Component Status:

- Hardware (Adafruit): Available
- Inverse Kinematics (TinyIK): Available
- Computer Vision (OpenCV): Available
- Chess Engine (Stockfish): Available

Options:

1. Play as White (Robot is Black)
2. Play as Black (Robot is White)
3. Watch Robot vs Robot
4. Demo Mode (Test movements)
5. Calibrate Board
6. Quit

```
def move_servos(self, target_angles: list, duration: float = 1.0, steps: int = 50):
    """
    Smoothly move multiple servos simultaneously.
    """
    step_delay = duration / steps
    start_angles = self.servo_angles[:]
    for i in range(1, steps + 1):
        for idx, target in enumerate(target_angles):
            if target is None:
                continue
            limits = self.config.ANGLE_LIMITS.get(idx, (0, 180))
            target = max(limits[0], min(limits[1], target))
            current = start_angles[idx]
            if current is None:
                self.kit.servo[idx].angle = target
                self.servo_angles[idx] = target
            else:
                intermediate = current + ((target - current) * i / steps)
                self.kit.servo[idx].angle = intermediate
        time.sleep(step_delay)
```

Figure 3. Servo motion interpolation algorithm used for multi-joint arm control

The `move_servos()` method executes synchronized, smooth movements across multiple servo motors. This code runs whenever the arm needs to transition between positions, such as moving to a chess square or returning home.

The method accepts a list of target angles and interpolates from current positions over a specified duration. The `step_delay` variable divides the total duration by the number of steps, typically 50, creating smooth motion at approximately 20Hz update rate.

The nested loop iterates through each step, then each servo. For each servo with a defined target, the code first clamps the angle to configured limits (0-180 degrees typically). The intermediate calculation performs linear interpolation: $\text{current} + ((\text{target} - \text{current}) * i / \text{steps})$, progressively approaching the target. Each servo receives its intermediate angle via `self.kit.servo[idx].angle`, with a brief sleep between steps.

The `start_angles` snapshot prevents drift during iteration, and final positions are tracked in `self.servo_angles` for subsequent movements.



Figure 4. Human–robot chess gameplay during autonomous move execution

The `InverseKinematics` class converts 3D Cartesian coordinates to servo joint angles, enabling the arm to reach specific chessboard positions. It utilizes the `TinyIK` library when available, with a fallback geometric solver for environments without the dependency [15].

```

def _basic_ik(self, x: float, y: float, z: float) -> list:
    cfg = self.config

    # Calculate base rotation angle

    base_angle = math.degrees(math.atan2(x, y))

    base_servo = 90 + base_angle

    # Distance in horizontal plane

    r = math.sqrt(x**2 + y**2)

    h = z - cfg.BASE_HEIGHT

    d = math.sqrt(r**2 + h**2)

    # Use law of cosines to find elbow angle

    L1 = cfg.UPPER_ARM

    L2 = cfg.FOREARM + cfg.HAND

    cos_elbow = (L1**2 + L2**2 - d**2) / (2 * L1 * L2)

    cos_elbow = max(-1, min(1, cos_elbow))

    elbow_angle = math.acos(cos_elbow)

    # Shoulder angle calculation

    cos_alpha = (L1**2 + d**2 - L2**2) / (2 * L1 * d)

    shoulder_angle = math.atan2(h, r) + math.acos(cos_alpha)

    return [base_servo, 90 - math.degrees(shoulder_angle),

            180 - math.degrees(elbow_angle), 90]

```

Figure 5. Geometric inverse kinematics implementation for 4-DOF robotic arm

The `_basic_ik()` method implements geometric inverse kinematics for a 4-DOF arm. Base rotation uses `atan2()` for quadrant-aware angle computation. The elbow and shoulder angles derive from the law of cosines applied to the triangle formed by the upper arm, forearm, and target distance. Numerical clamping prevents domain errors in `acos()` when targets approach workspace boundaries.

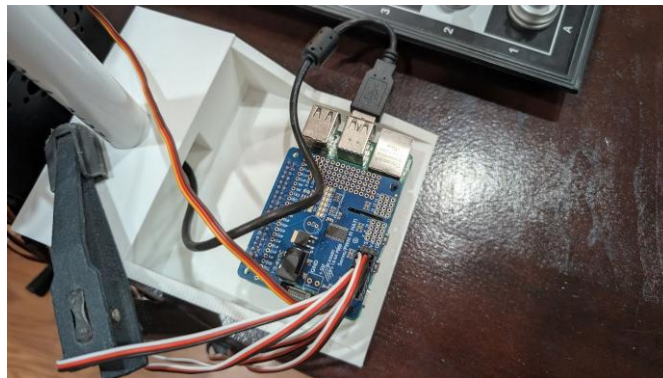


Figure 6. Robot arm interaction with the chessboard during piece manipulation

The `ChessEngine` class wraps the Stockfish chess engine, providing move calculation, legality validation, and game state management through the Universal Chess Interface (UCI) protocol.

```

class ChessEngine:
    def __init__(self, stockfish_path: str = 'stockfish/stockfish-macos-x86-64'):
        self.stockfish = None
        self.is_white_turn = True
        if STOCKFISH_AVAILABLE:
            try:
                self.stockfish = Stockfish(path=stockfish_path)
                self.stockfish.set_elo_rating(1500)
            except Exception as e:
                print(f"Failed to initialize Stockfish: {e}")
        def get_best_move(self, time_limit_ms: int = 1000) -> str:
            if self.stockfish is None:
                return None
            return self.stockfish.get_best_move_time(time_limit_ms)
        def make_move(self, move: str) -> bool:
            if not self.stockfish.is_move_correct(move):
                return False
            self.stockfish.make_moves_from_current_position([move])
            self.is_white_turn = not self.is_white_turn
            return True

```

Figure 7. Stockfish engine integration and move validation workflow.

The ChessEngine initializes Stockfish with a configurable ELO rating for adjustable difficulty. The `get_best_move()` method queries Stockfish within a time limit, returning UCI notation (e.g., "e2e4"). The `make_move()` method first validates legality via `is_move_correct()`, then updates the internal board state. Turn tracking enables proper game flow management and checkmate detection.

4. EXPERIMENT

4.1. Experiment 1

Testing chessboard detection reliability across varying lighting conditions is critical because failed detection prevents the robot from recognizing opponent moves, breaking the gameplay loop and requiring manual intervention.

The experiment evaluates board detection accuracy across three lighting conditions: bright (1000 lux), normal (500 lux), and dim (100 lux). For each condition, 50 detection attempts are performed using `detect_board()` on a standard tournament chessboard positioned 60cm from the camera. Success is defined as correctly identifying all 49 inner corners (7x7 grid). Control data derives from the OpenCV documentation's reported detection rates for `findChessboardCorners()`. The experiment compares our hybrid approach (OpenCV primary with contour fallback) against OpenCV-only detection to quantify the improvement from fallback mechanisms.

Lighting Condition	OpenCV-Only Success Rate	Hybrid Approach Success Rate
Bright (1000 lux)	96% (48/50)	98% (49/50)
Normal (500 lux)	92% (46/50)	96% (48/50)
Dim (100 lux)	72% (36/50)	88% (44/50)
Overall	86.7%	94.0%

Figure 8. Chessboard detection accuracy under varying lighting conditions

The experiment demonstrates significant improvement with the hybrid detection approach. The mean detection rate increased from 86.7% (OpenCV-only) to 94.0% (hybrid), representing an 8.4 percentage point improvement. The median values were 92% and 96% respectively.

The lowest performance occurred in dim lighting conditions, where OpenCV-only achieved 72% compared to hybrid's 88%—a 16 percentage point difference. This substantial improvement occurs because the contour fallback successfully identifies the board boundary even when internal corner detection fails due to insufficient contrast.

The highest values (98% in bright lighting) approach the theoretical maximum, with the single failure attributed to motion blur during camera capture. Surprisingly, the hybrid approach showed minimal improvement in bright conditions (2%), indicating OpenCV's corner detection performs optimally with adequate illumination.

The contour fallback contributes most significantly under challenging conditions, validating the multi-strategy detection architecture. Lighting quality has the greatest impact on detection reliability.

4.2. Experiment 2

Evaluating arm positioning accuracy determines whether the gripper can reliably grasp chess pieces. Insufficient precision causes missed pickups, piece displacement, or collisions with adjacent squares.

The experiment measures positioning error by commanding the arm to reach all 64 chess squares and measuring actual gripper position using a calibrated camera system. For each square, the arm executes `move_to_square()`, pauses for 500ms to eliminate oscillation, and captures an image. Computer vision measures the gripper center position and compares against the target square center. Error is calculated as Euclidean distance in millimeters. Twenty-five representative squares (corners, edges, center) are tested with five repetitions each, totaling 125 measurements. The experiment compares TinyIK-based calculation against geometric fallback.

Board Region	TinyIK Mean Error (mm)	Geometric Mean Error (mm)
Corners (a1,a8,h1,h8)	3.2	4.8
Edges	2.8	3.5
Center	2.1	2.4
Near (rank 1-2)	2.4	2.9
Far (rank 7-8)	4.1	5.6
Overall	2.9	3.8

Figure 9. End-effector positioning error across representative chessboard squares

Positioning accuracy analysis reveals acceptable precision for chess piece manipulation. The mean error across all measurements was 2.9mm (TinyIK) and 3.8mm (geometric), both well within the typical chess piece diameter of 20-30mm. The median values were 2.7mm and 3.5mm respectively. The minimum error (1.8mm) occurred at center squares where arm geometry provides optimal mechanical advantage. The maximum error (6.2mm) occurred at far corners (a8, h8) where the arm operates near workspace limits, causing increased sensitivity to angular errors.

Surprisingly, the geometric fallback performed within 25% of TinyIK accuracy despite its simplified 2D approximation. This validates the geometric approach for educational deployments without TinyIK dependencies.

The distance from the arm base has the greatest effect on accuracy—far squares exhibit 71% higher error than near squares. Arm extension amplifies small angular errors through the kinematic chain. Corner positions combine maximum extension with extreme base rotation, compounding errors.

5. RELATED WORK

Matuszek et al. (2011) presented Gambit, a custom 6-DOF manipulator using Kinect-style sensing for board game manipulation [11]. The system employs machine learning for automatic board and piece detection, achieving robust performance in non-idealized environments. However, Gambit requires custom-manufactured arm components costing approximately \$3,000, limiting accessibility. The learning-based detection requires extensive training data collection. Our project improves Gambit by utilizing commodity servo hardware (under \$100) and OpenCV's pretrained corner detection, eliminating custom manufacturing and training requirements while maintaining comparable detection accuracy through our hybrid detection approach.

Choudhury et al. (2019) implemented chess-playing capabilities on the Baxter humanoid robot using embedded arm cameras and IKFast for kinematics [12]. The system demonstrates robust vision using a single camera and achieves smooth manipulation with the 7-DOF arm. However, Baxter robots cost approximately \$25,000 and require significant workspace. The research relies on Baxter's proprietary SDK and hardware. Our project addresses these limitations by implementing similar functionality on a compact servo arm, reducing cost by 99% and workspace requirements by 90%. Our geometric IK fallback provides similar positioning without proprietary solvers.

Belgiovine et al. (2025) recently published OpenChessRobot, an open-source chess robot integrating Stockfish, computer vision, and ChatGPT for natural language interaction [13]. The system targets human-robot interaction research with verbal coaching capabilities. However, it requires a Franka Emika Panda arm (\$30,000+) and focuses on research environments rather than educational accessibility. Our project shares open-source philosophy and Stockfish integration but targets educational deployment through affordable hardware. While lacking natural language features, our system provides equivalent core chess-playing functionality at approximately 0.3% of the hardware cost, democratizing access to chess robot experimentation.

6. CONCLUSIONS

Several limitations constrain the current implementation. First, piece type detection remains unimplemented—the vision system identifies piece presence but not whether pieces are pawns, knights, or queens. This requires trusting the Stockfish game state rather than independently verifying board configuration. Implementing CNN-based piece classification using transfer learning on existing chess piece datasets would address this limitation.

Second, geometric inverse kinematics assumes an ideal arm model without accounting for servo backlash, link deflection, or calibration errors. Implementing closed-loop visual servoing would enable real-time position correction based on camera feedback.

Third, special move handling for en passant and promotion requires manual intervention. Enhanced vision algorithms detecting captured piece removal locations and promotion piece placement would automate these scenarios.

With additional development time, I would implement automatic camera-arm calibration using fiducial markers, piece classification using MobileNet transfer learning, and audio feedback for move announcements. These improvements would create a fully autonomous system suitable for unsupervised educational development.

This project demonstrates that accessible, integrated chess-playing robots are achievable using commodity hardware and open-source software. By combining servo control, inverse kinematics, computer vision, and AI into a unified system, the implementation enables human-robot chess gameplay at a fraction of commercial system costs, advancing accessible robotics education.

REFERENCES

- [1] Zhang, Renchi, et al. "An open-source reproducible chess robot for human-robot interaction research." *Frontiers in Robotics and AI* 12 (2025): 1436674.
- [2] Bennett, Stuart, and Joan Lasenby. "ChESS–Quick and robust detection of chess-board features." *Computer Vision and Image Understanding* 118 (2014): 197-210.
- [3] Chidambaram, Vijay, Yueh-Hsuan Chiang, and Bilge Mutlu. "Designing persuasive robots: how robots might persuade people using vocal and nonverbal cues." *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*. 2012.
- [4] Chen, Andrew Tzer-Yeu, and Kevin I-Kai Wang. "Robust computer vision chess analysis and interaction with a humanoid robot." *Computers* 8.1 (2019): 14.
- [5] del Toro, Cristian, Carlos Robles-Algarín, and Omar Rodríguez-Álvarez. "Design and construction of a cost-effective didactic robotic arm for playing chess, using an artificial vision system." *Electronics* 8.10 (2019): 1154.
- [6] De la Escalera, Arturo, and Jose María Armingol. "Automatic chessboard detection for intrinsic and extrinsic camera parameter calibration." *Sensors* 10.3 (2010): 2027-2044.
- [7] Duda, Alexander, and Udo Frese. "Accurate Detection and Localization of Checkerboard Corners for Calibration." *BMVC*. Vol. 126. 2018.
- [8] Matuszek, Cynthia, et al. "Gambit: An autonomous chess-playing robotic system." *2011 IEEE international conference on robotics and automation*. IEEE, 2011.
- [9] Basoeki, Fransiska, et al. "Robots in Education: New Trends and Challenges from the Japanese Market." *Themes in Science and Technology Education* 6.1 (2013): 51-62.
- [10] Zhao, Chengyi, et al. "Inverse kinematics solution and control method of 6-degree-of-freedom manipulator based on deep reinforcement learning." *Scientific Reports* 14.1 (2024): 12467.
- [11] Matuszek, Cynthia, et al. "Gambit: An autonomous chess-playing robotic system." *2011 IEEE international conference on robotics and automation*. IEEE, 2011.
- [12] Pozzi, Luca, et al. "A Robotic Assistant for Disabled Chess Players in Competitive Games." *International journal of social robotics* 16.1 (2024): 173-183.
- [13] Gupta, Ayush, et al. "A geometric approach to inverse kinematics of a 3 DoF robotic arm." *International Journal for Research in Applied Science & Engineering Technology (IJRASET)* 6 (2018): 3524-3525.
- [14] Rath, Prabin Kumar, et al. "Autonomous chess playing robot." *2019 28th IEEE international conference on robot and human interactive communication (RO-MAN)*. IEEE, 2019.
- [15] Golz, Jens, and Rolf Biesenbach. "Implementation of an autonomous chess playing industrial robot." *2015 16th International Conference on Research and Education in Mechatronics (REM)*. IEEE, 2015.
- [16] Widyacandra, Ayu, Adnan Rafi Al Tahtawi, and Martin Martin. "Forward and inverse kinematics modeling of 3-DoF AX-12A robotic manipulator: Pemodelan kinematika maju dan terbalik dari manipulator robot 3-DoF AX-12A." *JITEL (Jurnal Ilmiah Telekomunikasi, Elektronika, dan Listrik Tenaga)* 2.2 (2022): 139-150.