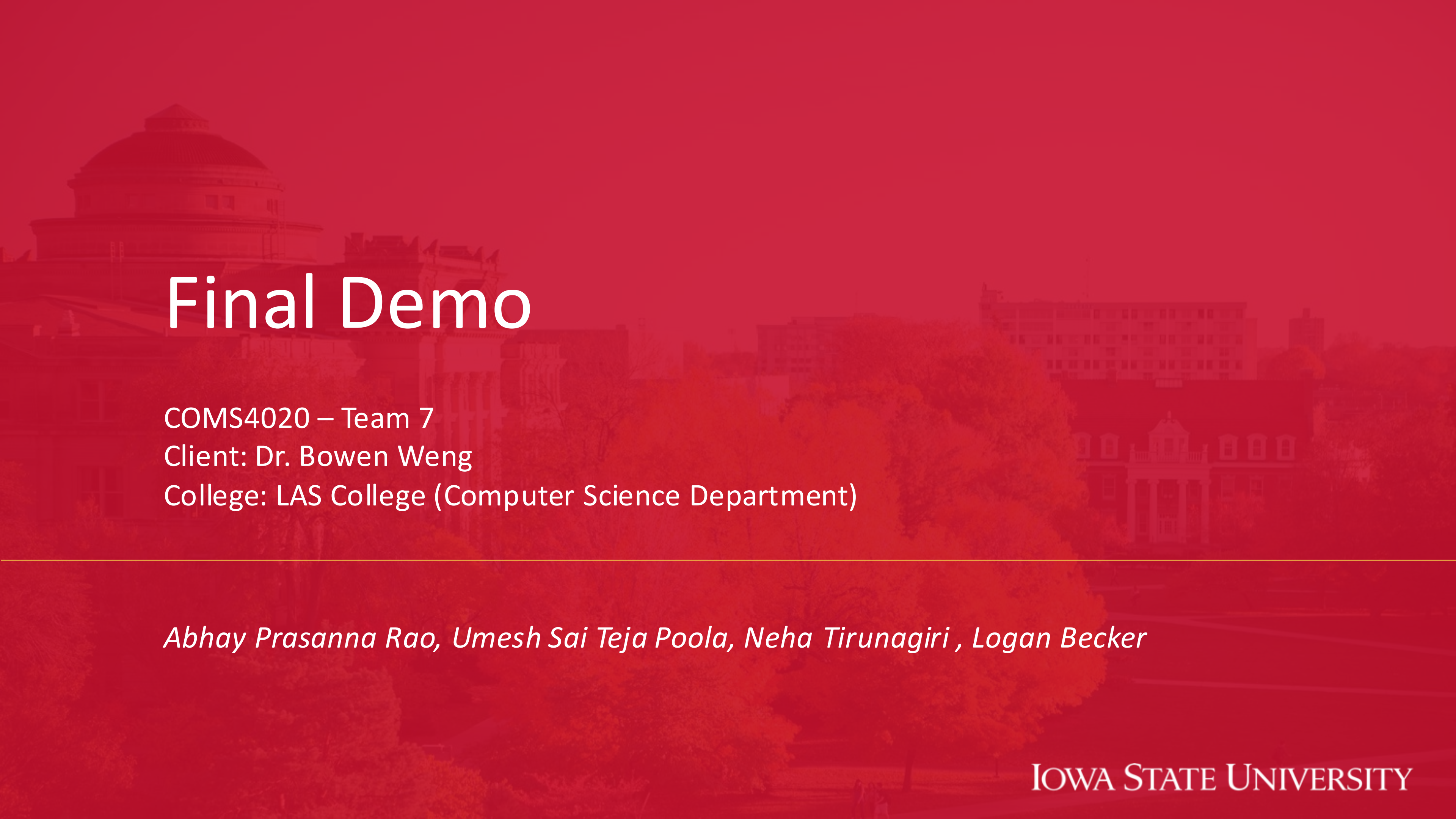
Iowa State University

COMS 4020 – Demo 3 Presentation

Team 7 – Robotic Chess Player



Final Demo

COMS4020 – Team 7
Client: Dr. Bowen Weng
College: LAS College (Computer Science Department)

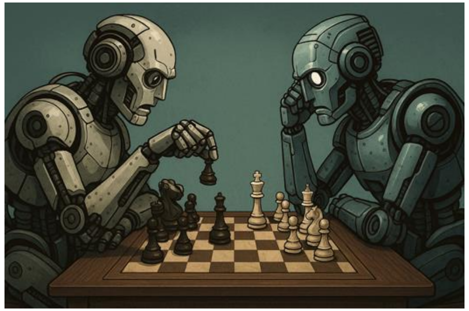
Abhay Prasanna Rao, Umesh Sai Teja Poola, Neha Tirunagiri , Logan Becker

IOWA STATE UNIVERSITY

Previous Demos Recap

Project Goal

- Build an autonomous robotic chess system that plays with real pieces.
- Integrate computer vision, chess AI, and robotic manipulation to remove the human operator.
- Enable a robot to perceive the board, compute moves, and execute them independently.



Why it Matters?

- Advances autonomous robotics and human - robot interaction.
- Demonstrates integration of vision, AI, and manipulation.
- Enables real-world deployment of chess AI without human intervention.

Solution

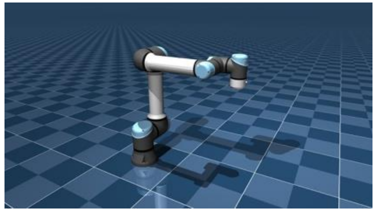
On a High Level, Here's what we are trying to accomplish:

1. Using a gripper hand attached to Robotic Arm (UR10e) to physically move the chess pieces during the gameplay
2. Using camera for Board Detection and Piece Recognition
3. Using Stockfish (Chess Engine) for moves calculation
4. Using Mujoco to simulate the robotic arm movements
5. Using ROS (Robotic Operation System) to bridge perception and action

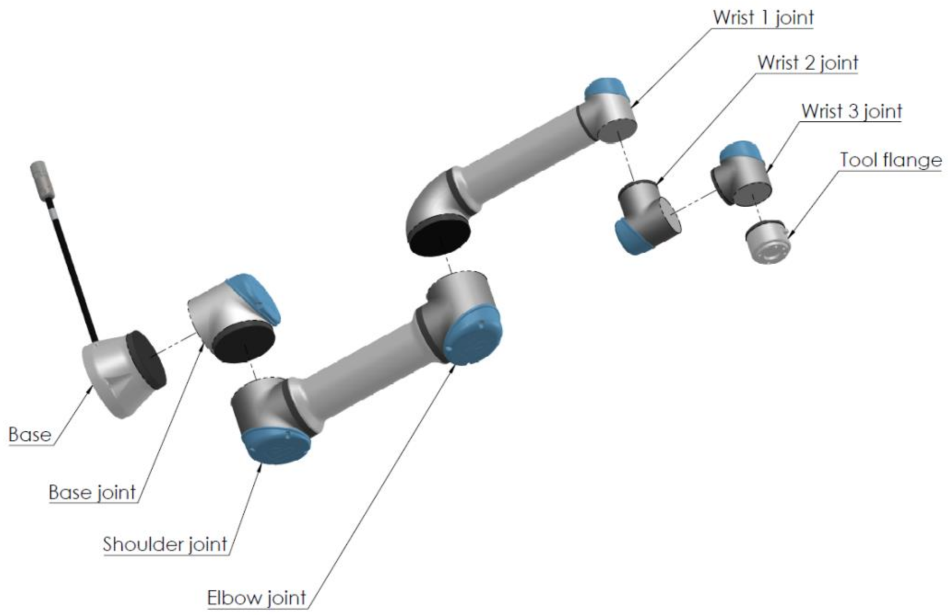
Hardware & Software

Software:
1. Stockfish (Chess Engine)
2. Yolo & OpenCV (Image recognition)
3. Yolo V8 nano (Piece recognition)
4. ROS (Robotic Operating System)
5. Mujoco
6. Python
7. Ubuntu OS+ Catkin Workspace

Hardware:
1. UR10e Robotic Arm
2. Hand Gripper
3. Logitech Webcam
4. Chess Board & Pieces



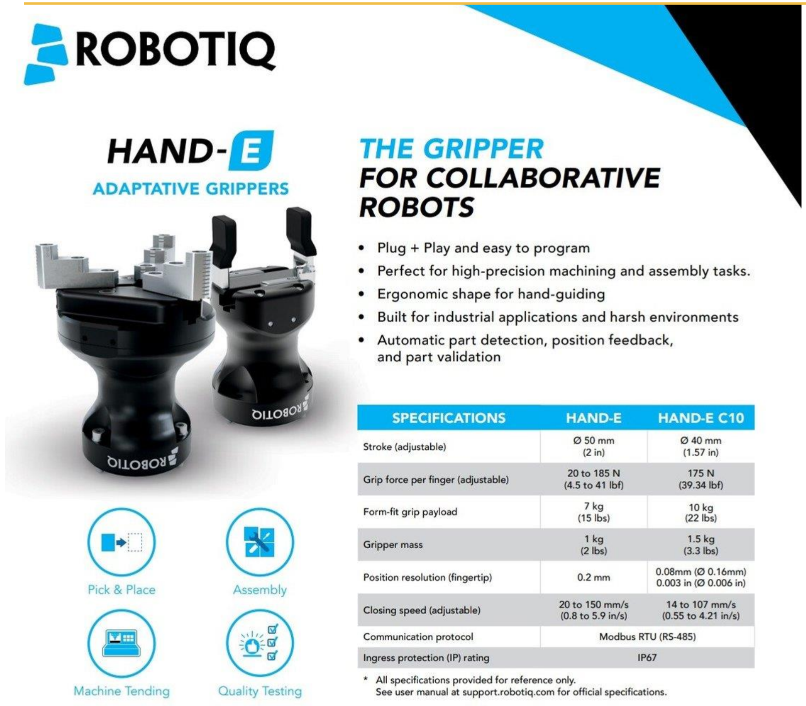
UR10e Robot Hardware



Universal Robot's UR10e

- 6 degrees of Freedom:
  - shoulder_pan_joint (base joint)
  - shoulder_lift_joint
  - elbow_joint
  - wrist_1_joint
  - wrist_2_joint
  - wrist_3_joint

Gripper Hardware



ROBOTIQ

HAND-E

ADAPTATIVE GRIPPERS

THE GRIPPER FOR COLLABORATIVE ROBOTS

- Plug + Play and easy to program
- Perfect for high-precision machining and assembly tasks.
- Ergonomic shape for hand-guiding
- Built for industrial applications and harsh environments
- Automatic part detection, position feedback, and part validation

SPECIFICATIONS	HAND-E	HAND-E C10
Stroke (adjustable)	Ø 50 mm (2 in)	Ø 40 mm (1.57 in)
Grip force per finger (adjustable)	20 to 185 N (4.5 to 41 lbf)	175 N (39.34 lbf)
Form-fit grip payload	7 kg (15 lbs)	10 kg (22 lbs)
Gripper mass	1 kg (2 lbs)	1.5 kg (3.3 lbs)
Position resolution (fingertip)	0.2 mm	0.08mm (Ø 0.16mm) 0.003 in (Ø 0.006 in)
Closing speed (adjustable)	20 to 150 mm/s (0.8 to 5.9 in/s)	14 to 107 mm/s (0.55 to 4.21 in/s)
Communication protocol	Modbus RTU (RS-485)	
Ingress protection (IP) rating	IP67	

* All specifications provided for reference only.
See user manual at support.robotiq.com for official specifications.

Pick & Place

Assembly

Machine Tending

Quality Testing

Robotiq **Hand-E** Adaptive Gripper:

- Dual adaptive fingers (2-inch span): conforms to varied part shapes

- Position-controlled interface: command open/close and read joint state for feedback

- Position-based grasp check: infer success from achieved position

Board Vision Pipeline

Camera → Corner Detection → Homography → Bird's Eye View → Grid Mapping → YOLO Piece Recognition → FEN Generation
1fps 7 Methods Validation 800×800px 64 Squares

Detection Methods

Combined Robust (Primary)

Canny edges, contour filtering, convex-hull corner extraction; includes an automatic fallback with 8 methods ranked.

YOLO + Contour Detection

Adaptive threshold, contours, approxPolyDP with ε = 0.02 × arcLength; validation for 4 corners, area > 10k px, multiple variants (Mean, Gaussian, CLAHE).

OpenCV Chessboard Detection

findChessboardCornersSB for various grid sizes (7×7, 6×6, 5×5, 8×8, 9×9), inner-grid to outer-corner extrapolation, subpixel refinement with cornerSubPix().

Contour Adaptive Fallback

CLAHE, Canny, morphological closing.

Edge-Based Fallback

Canny with dilation/erosion.

Harris Lines Fallback

Harris corners with convex hull.

Robustness to Real-World Disruptions

Challenges During Live Gameplay

- Camera Movement: Accidental bumps or tripod shifts change perspective.

- Board Movement: Robot arm or human interaction may shift the board.

- Robot-Induced Vibrations: Small shakes can distort corners or warp the board view.

- Pawns on the Board: Mid-game recalibration is difficult because pieces obstruct corners.

- Camera Dust / Clarity Issues: Detection confidence drops when the lens is partially blocked.

Saved Calibration

- The system stores a clean, pre-game calibration before pawns are placed.

User-Guided Corner Mapping

- The operator can press k to manually select the 4 board corners.

- Instantly rebuilds homography

- Avoids removing pieces

- Rescues the game without restarting

Fallback When Auto-Detection Fails

- If mid-game the view shifts and automatic methods fail due to piece interference or noise...

Real-Time Recalibration Without Restart

- Real-time recalibration must still be possible without restarting the match, ensuring continuous gameplay.

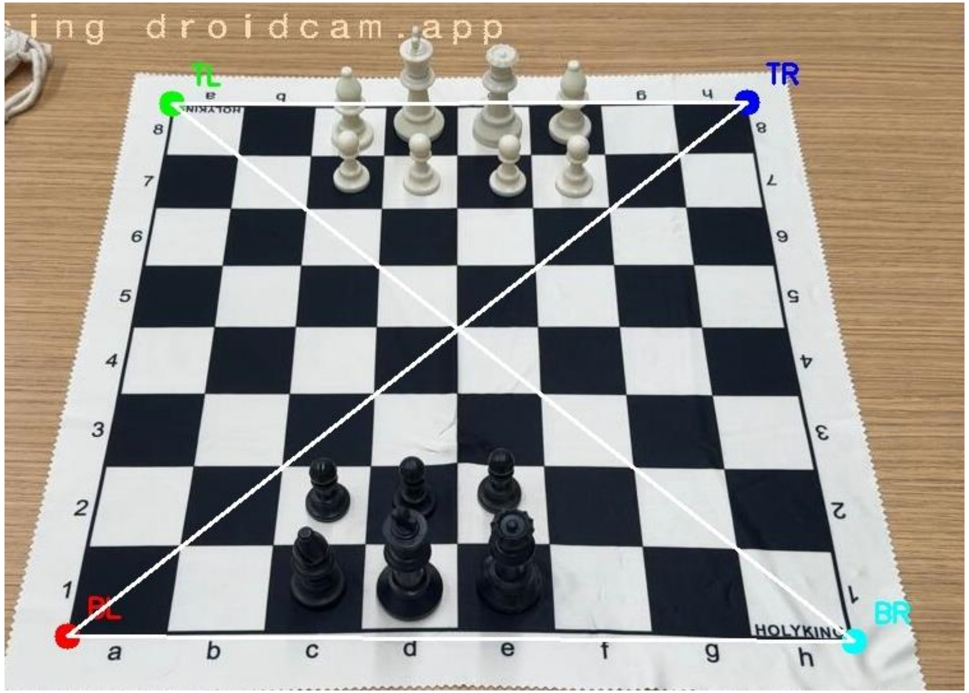
System Stability

- The system remains stable, recoverable, and fully playable even under real-world disturbances.

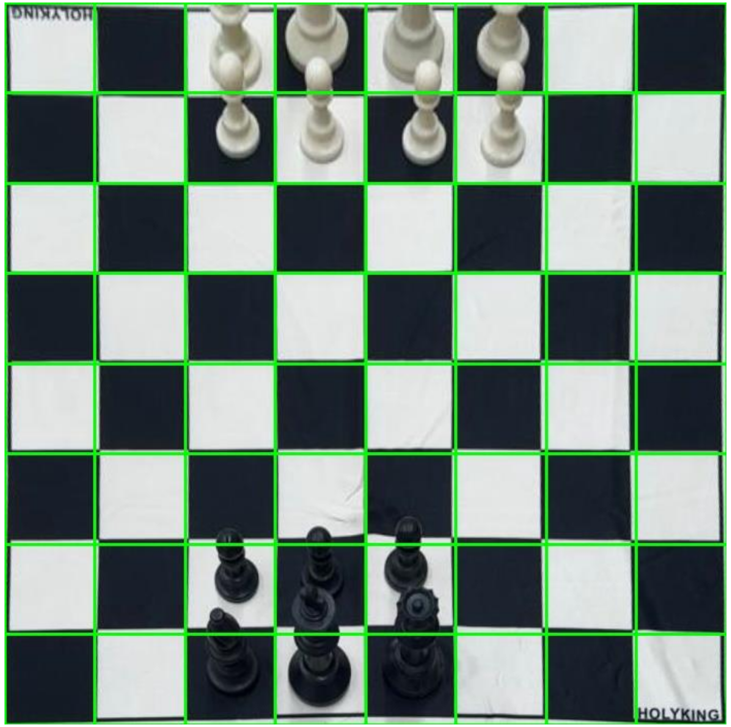
Piece Recognition – Training and Stuff

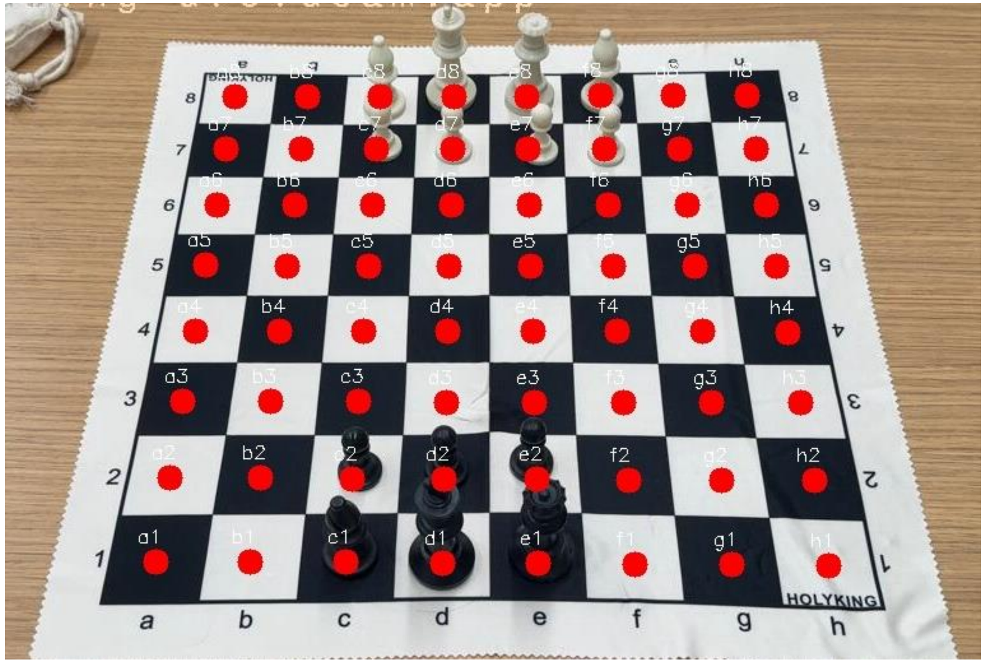
•Our system is designed with **multiple fallback detection methods** to ensure robustness under real-world conditions.
•We begin with **YOLO-based object detection** to identify chess pieces and board regions efficiently.
•In parallel, **contour detection** is used to validate piece boundaries and handle cases where confidence scores fluctuate.
•To improve structural accuracy, we apply **gradient-based edge detection** followed by **Hough transforms** to detect board lines reliably.
•For precise board geometry, **OpenCV chessboard corner detection** is used to localize square corners accurately.
•This **layered and redundant vision pipeline** allows the system to remain stable despite lighting changes, partial occlusions, or camera angle variations.

Live Detection With Pieces on the Board (Corner)

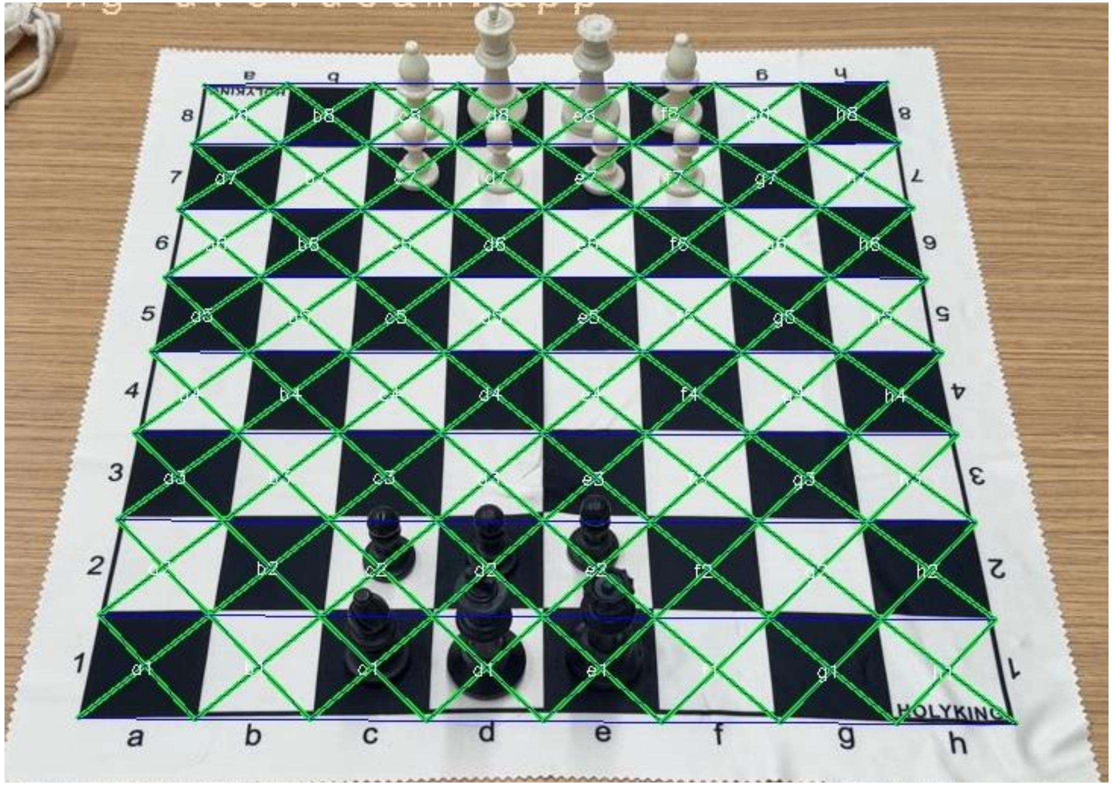


Live Detection With Pieces on the Board (BirdsEye + Square Mapping)

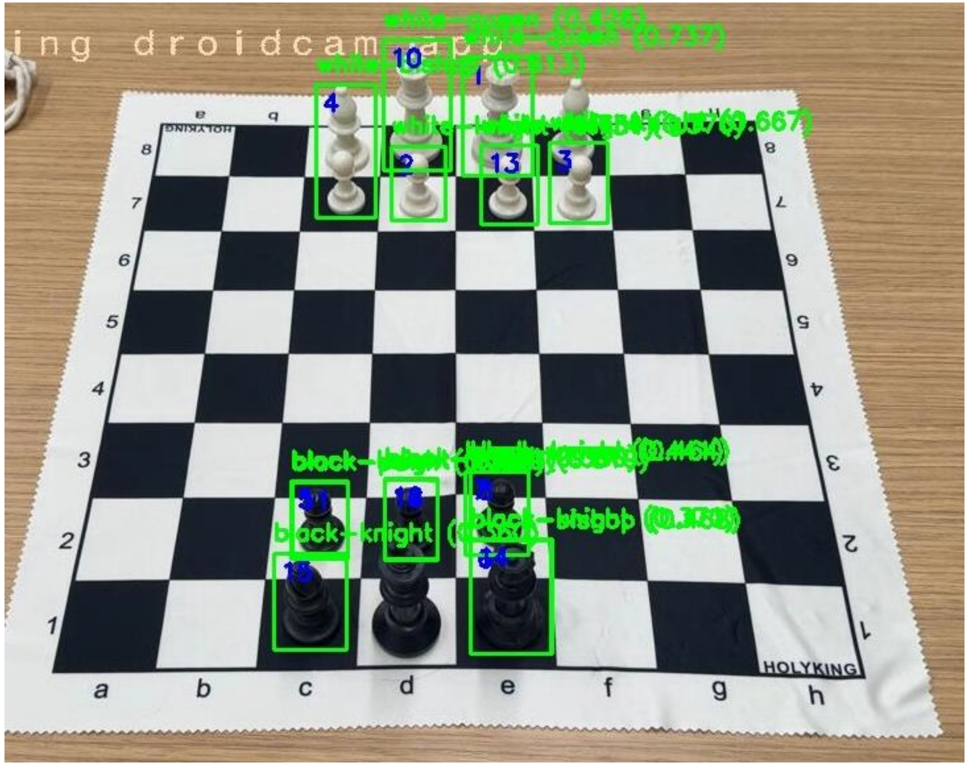




Live Detection With Pieces on the Board (Diagonal Mapping)



Live Detection With Pieces on the Board (YOLO Piece Recognition)

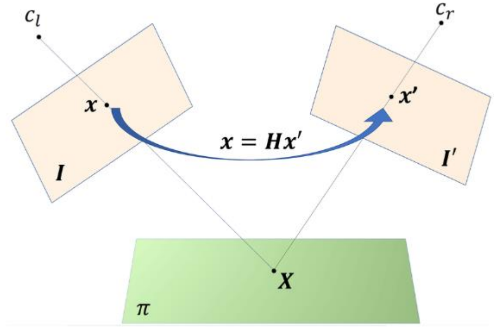


Homographic Transformation

What is a Homography?

- A 3×3 perspective matrix that maps points from the camera image to a top-down (bird's-eye) chessboard view.
- Used to remove perspective distortion and align the board to an 8×8 grid.

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad (x_{final}, y_{final}) = \left(\frac{x'}{w'}, \frac{y'}{w'} \right)$$



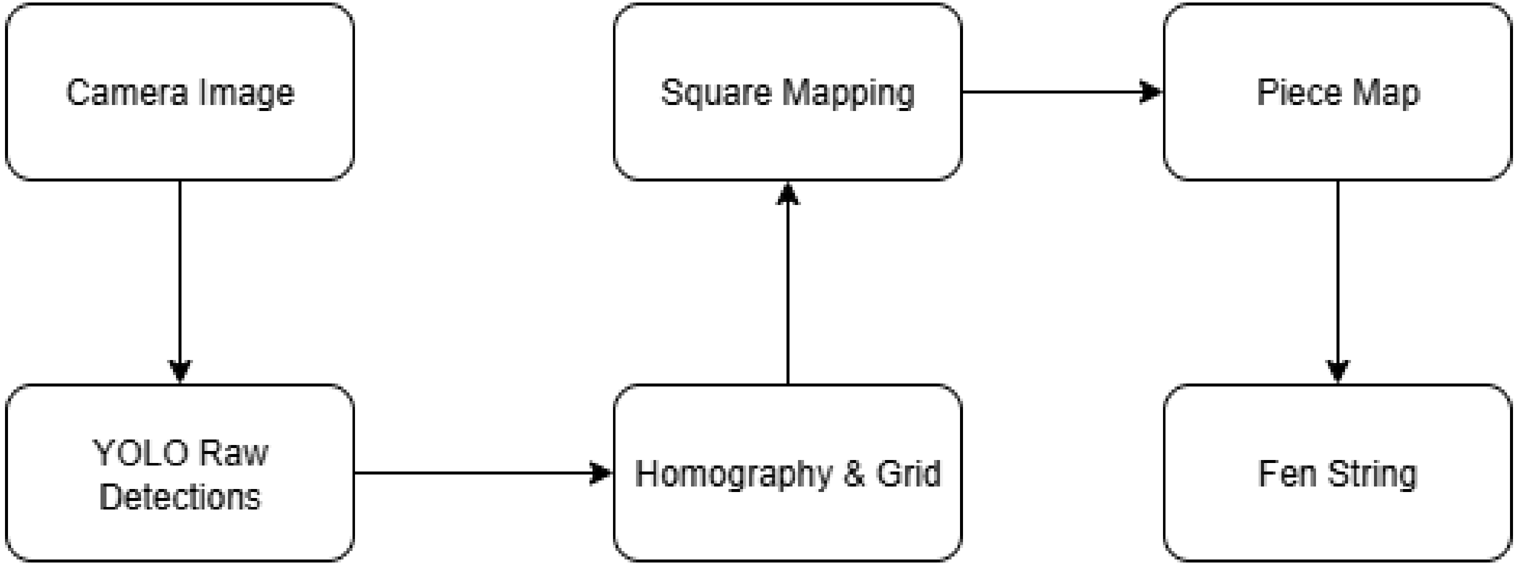
Core Operations

- **Compute H**: cv2.getPerspectiveTransform(src_corners, dst_corners)
- **Warp Image**: cv2.warpPerspective(image, H, (800, 800))
- **Transform Points**: cv2.perspectiveTransform(points, H)
- **Inverse Mapping**: cv2.perspectiveTransform(points, H_inv)

Why We Need It

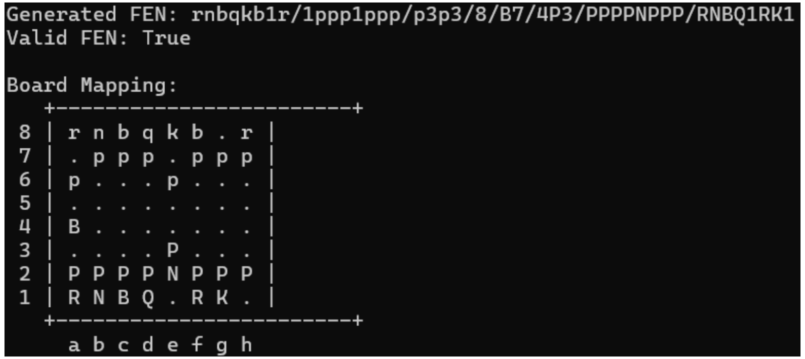
- Converts raw camera detections into **consistent board coordinates**.
- Enables stable chess square mapping even if the camera angle changes.
- Allows YOLO piece positions to be mapped directly to **FEN board squares**.

FEN Generation Pipeline



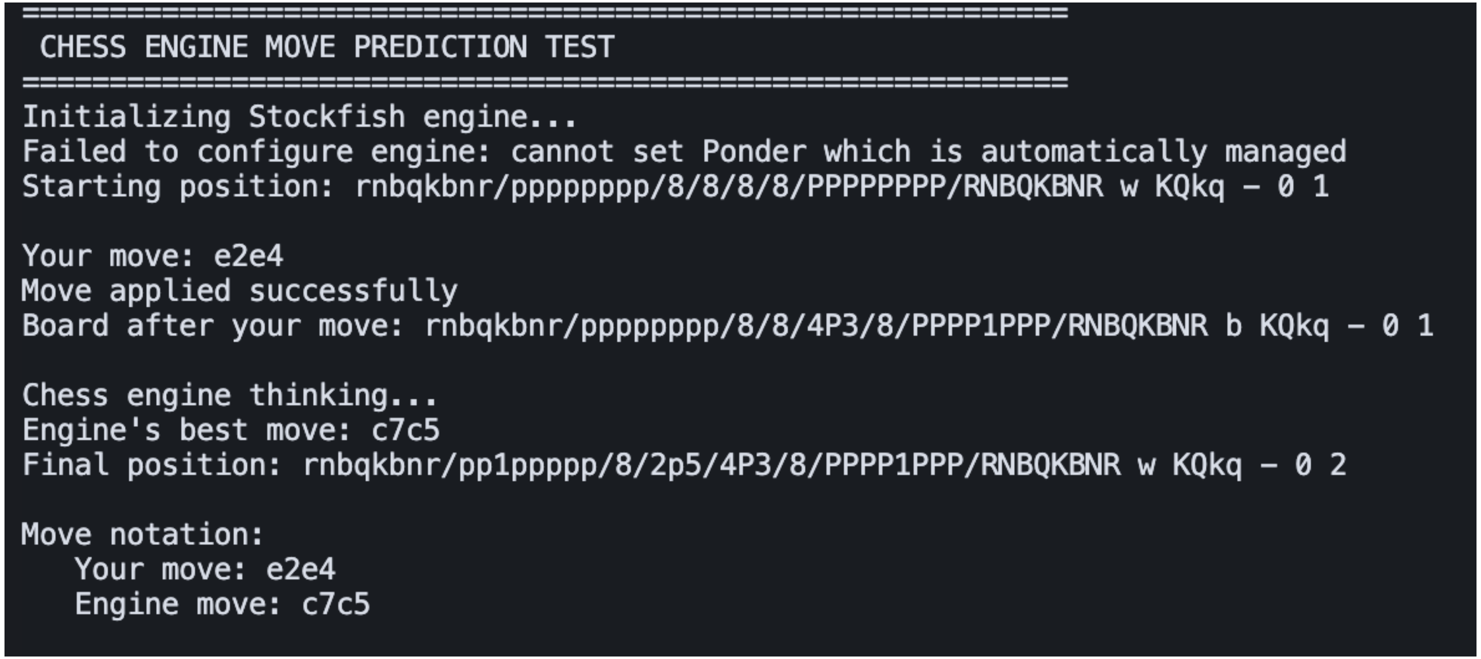
FEN Generation + Stockfish Input

- Get game state part of fen from web socket server (castling rights, fullmove and halfmove counter)
- Web socket combines piece placement fen with game logic fen
- Chess Manager queries the web socket for the current fen
- Stockfish finds the best move for the given fen



```
Generated FEN: rnbqkb1r/1ppp1ppp/p3p3/8/B7/4P3/PPPPNPPP/RNBQ1RK1
Valid FEN: True

Board Mapping:
   +-----------------------+
 8 | r n b q k b . r |
 7 | . p p p . p p p |
 6 | p . . . p . . . |
 5 | . . . . . . . . |
 4 | B . . . . . . . |
 3 | . . . . P . . . |
 2 | P P P P N P P P |
 1 | R N B Q . R K . |
   +-----------------------+
     a b c d e f g h
```

Chess Engine Setup & Best Move Prediction

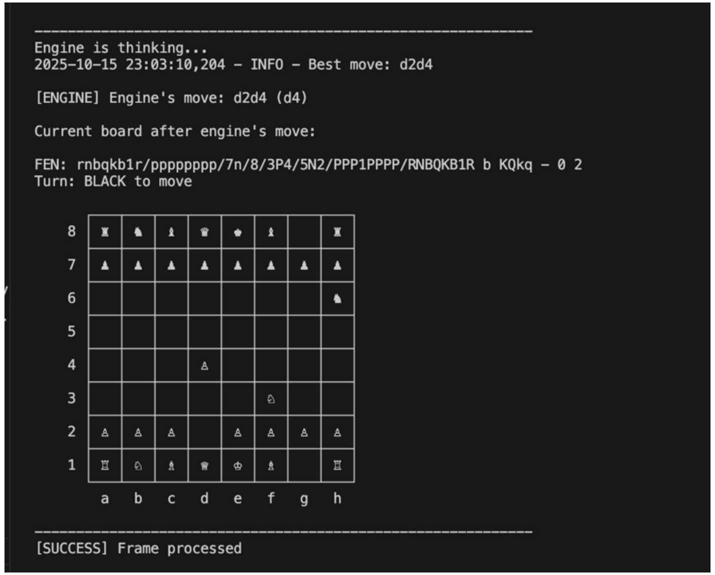


```
============================================================
 CHESS ENGINE MOVE PREDICTION TEST
============================================================
Initializing Stockfish engine...
Failed to configure engine: cannot set Ponder which is automatically managed
Starting position: rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1

Your move: e2e4
Move applied successfully
Board after your move: rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq - 0 1

Chess engine thinking...
Engine's best move: c7c5
Final position: rnbqkbnr/pp1ppppp/8/2p5/4P3/8/PPPP1PPP/RNBQKBNR w KQkq - 0 2

Move notation:
   Your move: e2e4
   Engine move: c7c5
```

Fall Back Approach - Blind Chess Game Logic

- The Blind Chess system lets users play chess **without manually entering moves**.
- It detects **piece movements through YOLO-based computer vision** and synchronizes with **Stockfish** for live gameplay.
- The system compares the detected board with the previous state to **infer the player's move**, then updates the board for Stockfish's response.

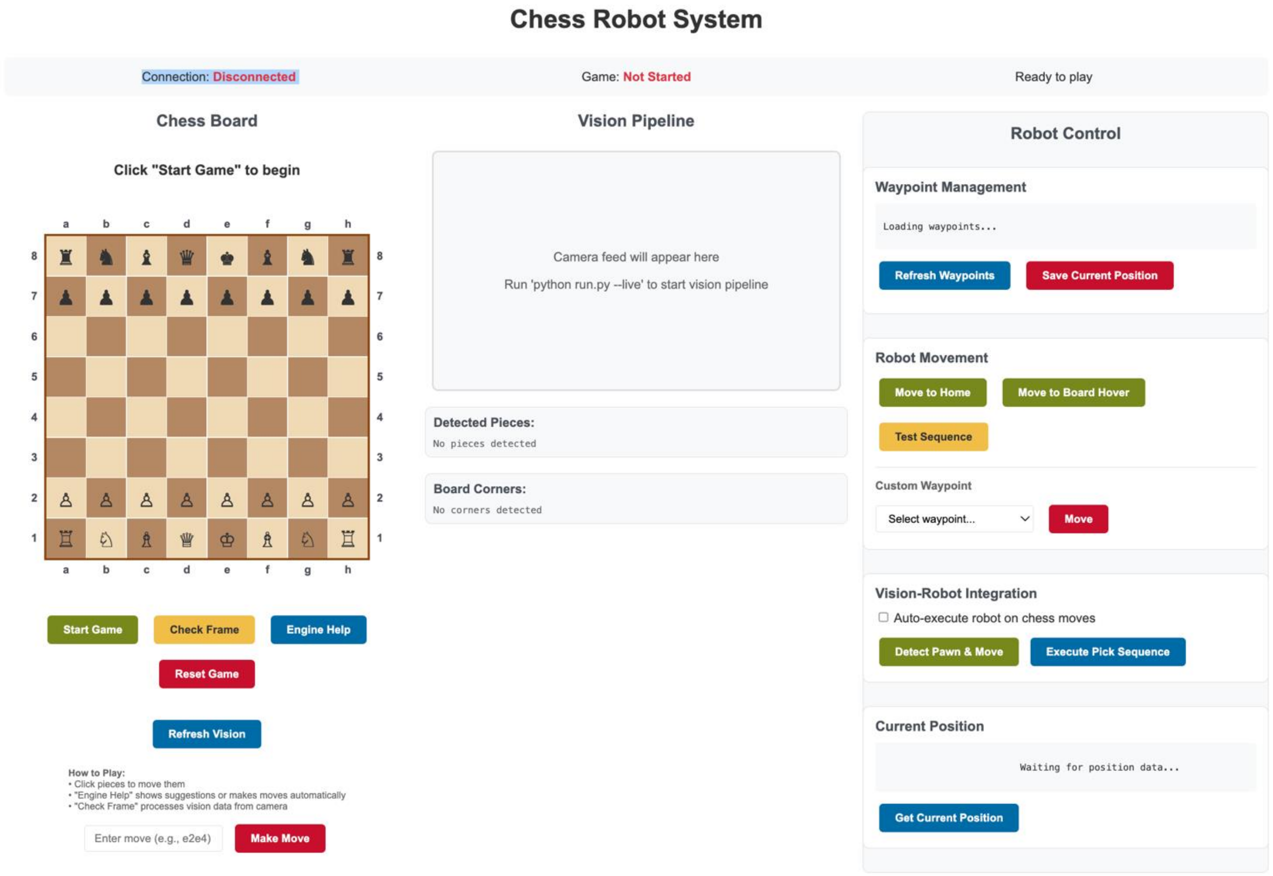
Camera → YOLO → FEN → Reconciliation → Stockfish → Updated Board



```
------------------------------------------------------------
Engine is thinking...
2025-10-15 23:03:10,204 - INFO - Best move: d2d4

[ENGINE] Engine's move: d2d4 (d4)

Current board after engine's move:

FEN: rnbqkb1r/pppppppp/7n/8/3P4/5N2/PPP1PPPP/RNBQKB1R b KQkq - 0 2
Turn: BLACK to move

  8 | ♜ | ♞ | ♝ | ♛ | ♚ | ♝ |   | ♜ |
  7 | ♟ | ♟ | ♟ | ♟ | ♟ | ♟ | ♟ | ♟ |
  6 |   |   |   |   |   |   |   | ♞ |
  5 |   |   |   |   |   |   |   |   |
  4 |   |   |   | ♙ |   |   |   |   |
  3 |   |   |   |   |   | ♘ |   |   |
  2 | ♙ | ♙ | ♙ |   | ♙ | ♙ | ♙ | ♙ |
  1 | ♖ | ♘ | ♗ | ♕ | ♔ | ♗ |   | ♖ |
      a   b   c   d   e   f   g   h

------------------------------------------------------------
[SUCCESS] Frame processed
```

Web Integration for Game Controls

- **Interactive Web Interface**
    - Users make moves directly on the digital chessboard via the web interface.
- **Move Execution**
    - When a user makes a move, the **UR10e robot arm** executes the corresponding pick-and-place sequence through the robot movement logic.
- **Live Video Display**
    - A real-time camera feed is displayed in the web interface to show the actual board state during gameplay, complementing simulation views.
- **AI Response**
    - The **Stockfish** engine calculates the robot's move, updates the board state, and triggers the robotic arm to move the piece.
- **Webapp Integration**
    - Full system integration is complete: **Vision + Stockfish + Webapp + ROS**.


Chess Robot System

Connection: Disconnected | Game: Not Started | Ready to play

Chess Board

Click "Start Game" to begin

How to Play:
- Click pieces to move them
- "Engine Help" shows suggestions or makes moves automatically
- "Check Frame" processes vision data from camera

Vision Pipeline

Camera feed will appear here

Run 'python run.py --live' to start vision pipeline

Detected Pieces:

```
No pieces detected
```

Board Corners:

```
No corners detected
```

Robot Control

Waypoint Management

```
Loading waypoints...
```

Robot Movement

Custom Waypoint

Vision-Robot Integration

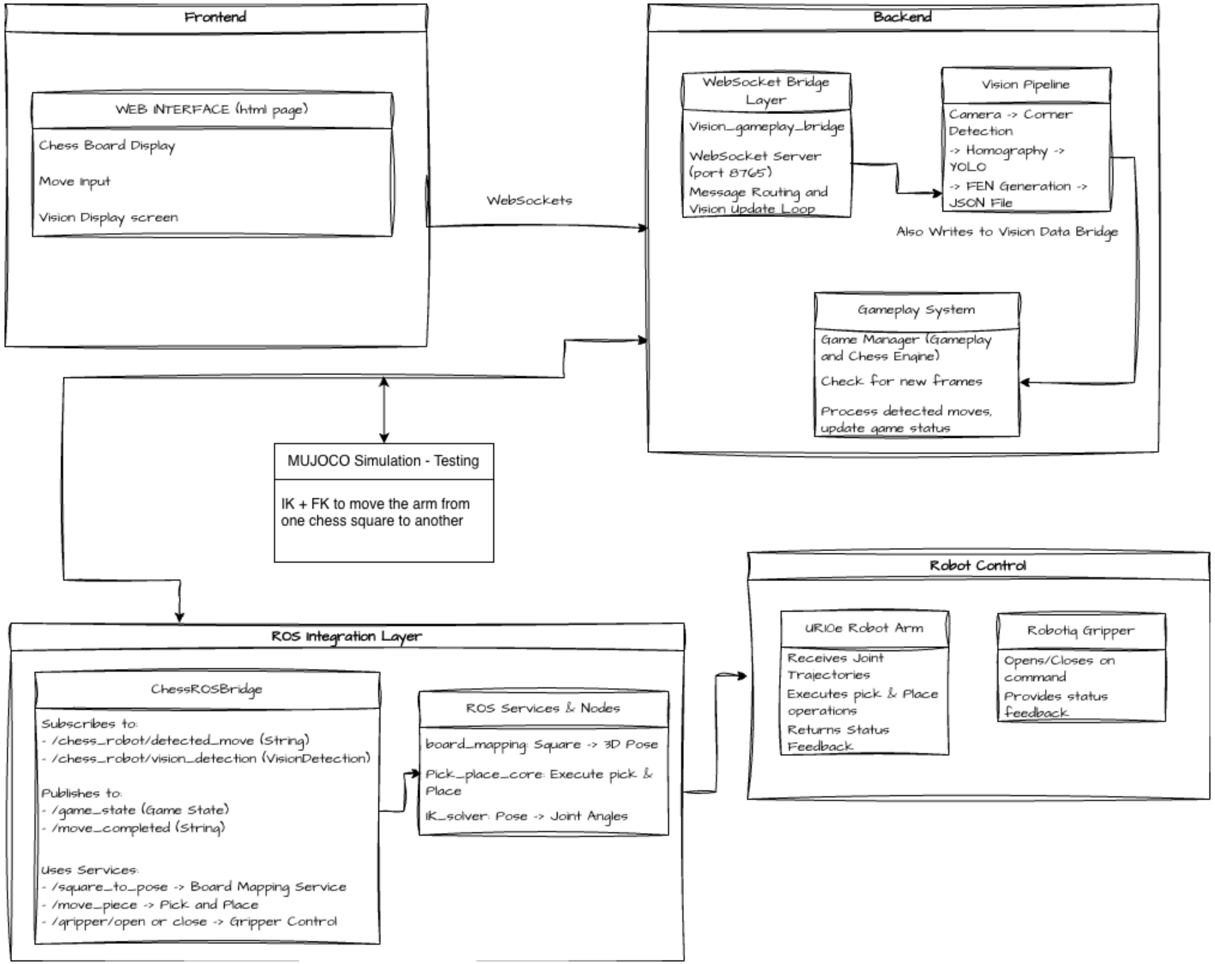
☐ Auto-execute robot on chess moves

Current Position

```
Waiting for position data...
```

Placeholder, will re-update it in the Lab tomorrow at 3pm

System Architecture


Frontend
WEB INTERFACE (html page)
Chess Board Display
Move Input
Vision Display screen
WebSockets
Backend
WebSocket Bridge Layer
Vision_gameplay_bridge
WebSocket Server (port 8765)
Message Routing and Vision Update Loop
Vision Pipeline
Camera -> Corner Detection
-> Homography -> YOLO
-> FEN Generation -> JSON File
Also Writes to Vision Data Bridge
Gameplay System
Game Manager (Gameplay and Chess Engine)
Check for new frames
Process detected moves, update game status
MUJOCO Simulation - Testing
IK + FK to move the arm from one chess square to another
Robot Control
UR10e Robot Arm
Receives Joint Trajectories
Executes pick & Place operations
Returns Status Feedback
Robotiq Gripper
Opens/Closes on command
Provides status feedback
ROS Integration Layer
ChessROSBridge
Subscribes to:
- /chess_robot/detected_move (String)
- /chess_robot/vision_detection (VisionDetection)
Publishes to:
- /game_state (Game State)
- /move_completed (String)
Uses Services:
- /square_to_pose -> Board Mapping Service
- /move_piece -> Pick and Place
- /gripper/open or close -> Gripper Control
ROS Services & Nodes
board_mapping: Square -> 3D Pose
Pick_place_core: Execute pick & Place
IK_solver: Pose -> Joint Angles

Building a Robot Command

System Architecture Overview

Kinematics Solver

Forward Kinematics (FK)

In: Robot state ($\theta_1, \theta_2 \ldots \theta_6$)

Out: Gripper position (x, y, z)

Direct calculation using DH parameters

⇅

Inverse Kinematics (IK)

In: Target position (x, y, z)

Out: Robot state ($\theta_1, \theta_2 \ldots \theta_6$)

Solved via Levenberg-Marquardt

→

GameManager

desired action

↓

Controller

In: Chess action (move, capture, etc.)

Out: Sequence of waypoints

Translates high-level commands to poses

↓

Waypoint Table

Maps board positions to joint angles

Pre-calibrated lookup for speed

robot joint trajectories

↓

ROS Goals

Forward Kinematics

Computing End-Effector Position from Joint Angles

Key Concept: Treat each physical part of the robot as a *transformation*

- Links are positional offsets
- Joints are rotations

Mathematical Framework:

- Transformations belong to Special Euclidean group *SE(3)* of **rigid motions**
- **Net effect:** brings origin to end position

Overall transformation = product of individual matrices

Transformation Chain:

$$\begin{tikzcd}
\textbf{base} \arrow[r] & \textbf{joint 1} \arrow[r] \arrow[d] & \textbf{link 2} \arrow[r] \arrow[d] & \textbf{joint 3} \arrow[r] \arrow[d] & \textbf{link 4} \arrow[r] \arrow[d] & \textbf{gripper} \\
\begin{array}{c}\textbf{origin}\\(0, 0, 0)\end{array} \arrow[r] & \begin{array}{c}\textbf{rotate by}\\\text{angle } \theta_1\end{array} \arrow[r] & \begin{array}{c}\textbf{offset by}\\\text{length } x_2\end{array} \arrow[r] & \begin{array}{c}\textbf{rotate by}\\\text{angle } \theta_3\end{array} \arrow[r] & \begin{array}{c}\textbf{offset by}\\\text{length } x_4\end{array} \arrow[r] & \begin{array}{c}\textbf{end}\\\text{position}\end{array}
\end{tikzcd}$$

Inverse Kinematics

Problem: Given a target 3D position, find the joint angles such that the gripper ends up at the target.

Key Challenges: Nonlinear problem due to rotational (revolute) joints

Optimization Approach: use a nonlinear optimizer from libraries (e.g. scipy)

Minimize cost function: the squared error from target $\bar{p}$

$$j(\bar{q}) = \tfrac{1}{2}\, ||FK(\bar{q}) - \bar{p}||^2$$

$$j(\bar{q}) = \tfrac{1}{2}\, ||FK(\bar{q}) - \bar{p}||^2$$

Where q is a tuple of 6 angles, in parameter space $\Theta \subseteq T^6$ with constraints and known bounds to prevent unsafe configurations:

$$IK(\bar{q}) = \arg\min j(\bar{q})$$

Levenberg-Marquardt Method: Combines Gauss-Newton and gradient descent

`

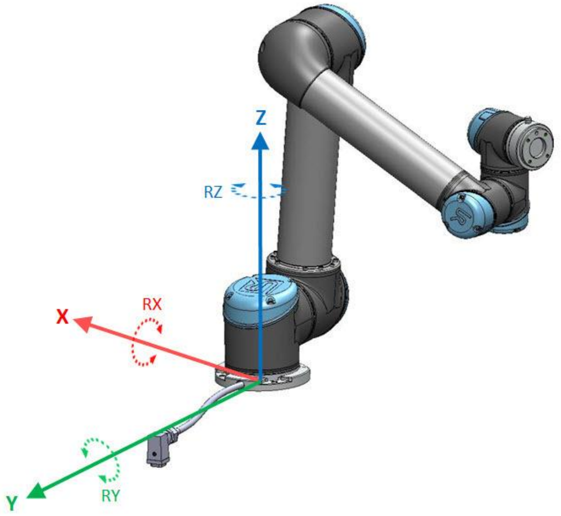
$$x = x + (JJ + \lambda I)\ Jr(x)$$

J = Jacobian, r(x) = residual vector, λ = damping factor

Waypoints and Calibration

- This is the next stage where we need to set some pre-calibrated waypoints to use for the robot's initial placements
- This will map physical positions to robot states
- We will set Home to be our start/end states
- Table position for captured pieces
- 2 waypoints for each square:
  - Square Hover: Directly above
  - Square Drop: drop/pick the piece at it's square

Totally we will have about 64x2 board and 2 custom waypoints



Subscribers and Publishers

Subscribers

(What our system listens to)

/joint_states
Receives real-time robot arm joint feedback for trajectory synchronization.

/chess_robot/vision_detection
Subscribes to YOLO-based vision detection outputs to interpret chess piece locations.

/chess_robot/detected_move
Subscribes to detected or user-triggered chess moves (UCI format).

Trajectory Execution Feedback
Monitors robot trajectory execution status via ROS **Action feedback**
(MoveIt-free system)

Publishers

(What our system sends out)

Robot Trajectory Actions
Sends joint trajectories via FollowJointTrajectoryAction to execute pick-and-place motions.

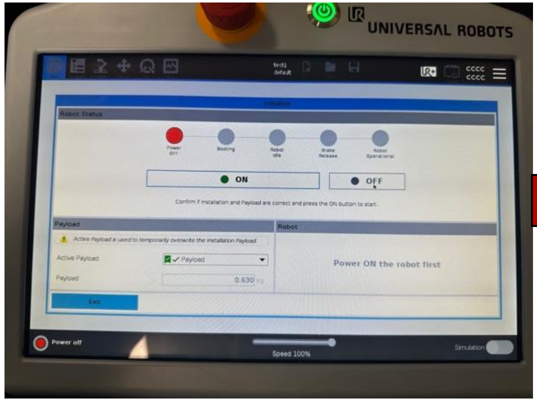
Gripper Commands
Sends open/close commands via ROS **Services** (/gripper/open, /gripper/close) *(currently tested manually due to gripper malfunction)*

/chess_robot/game_state
Publishes the detected chessboard configuration and full game state per frame.

/chess_robot/move_completed
Notifies when a move is completed or when fallback logic is required.

System Setup

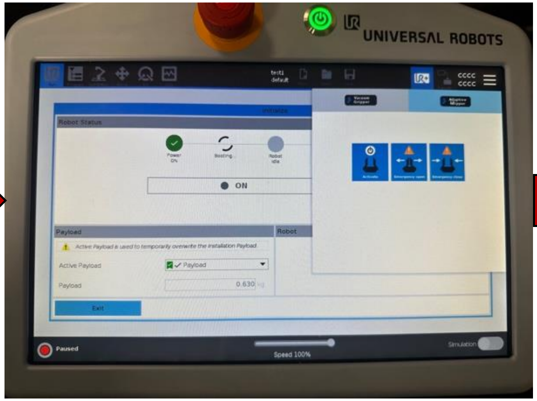
UR Robot Startup & Safety Initialization



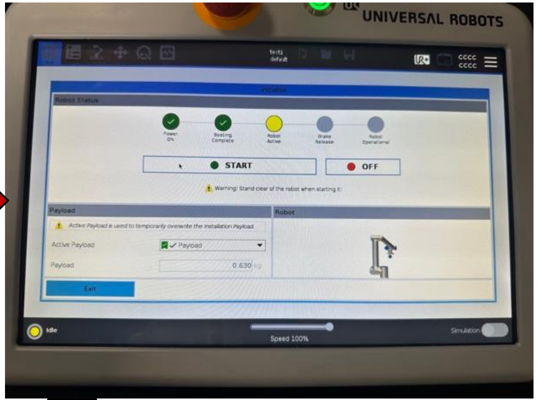
Confirm if installation and Payload are correct and press the ON button to start.

Active Payload is used to temporarily overwrite the installation Payload.

Power ON the robot first

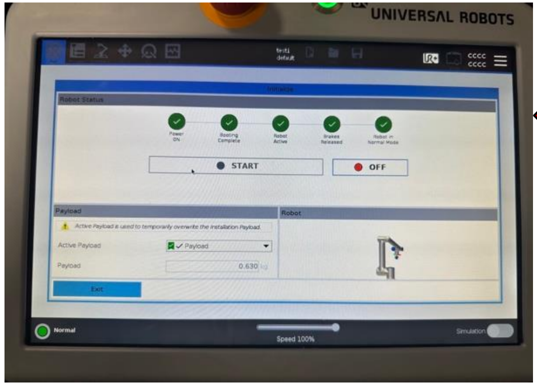


Active Payload is used to temporarily overwrite the installation Payload.



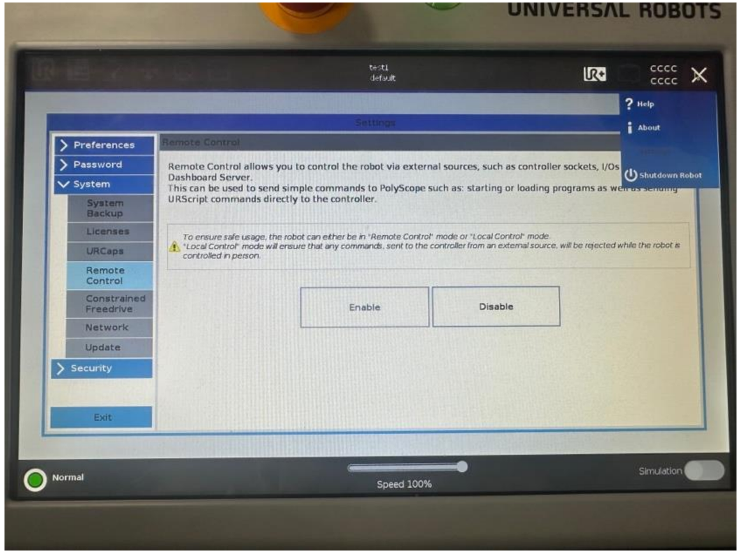
Warning: Stand clear of the robot when starting it!

Active Payload is used to temporarily overwrite the installation Payload.



Active Payload is used to temporarily overwrite the installation Payload.

Enabling Remote Control Mode

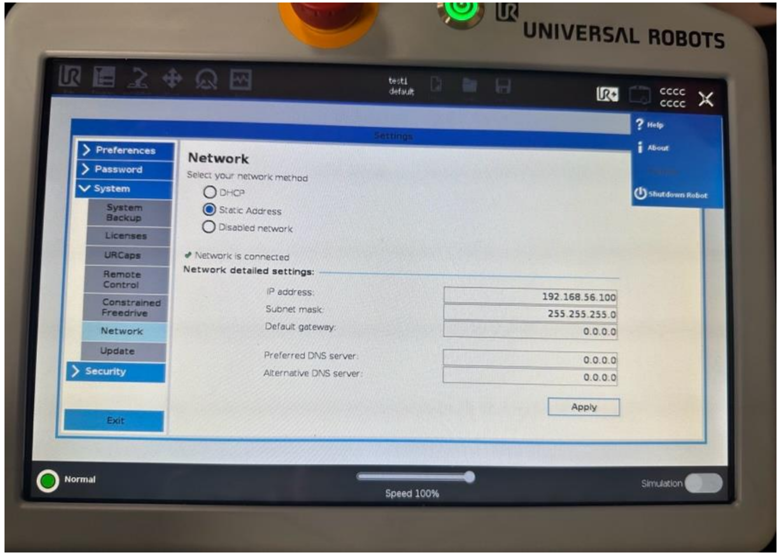


Remote Control allows you to control the robot via external sources, such as controller sockets, I/Os
Dashboard Server.
This can be used to send simple commands to PolyScope such as: starting or loading programs as we
URScript commands directly to the controller.

To ensure safe usage, the robot can either be in "Remote Control" mode or "Local Control" mode.
"Local Control" mode will ensure that any commands, sent to the controller from an external source, will be rejected while the robot is controlled in person.

- This setting allows external software (ROS + URCap ExternalControl) to send commands to the robot.
- Enable "Remote Control Mode" to allow program execution from the ROS pipeline.

Network Configuration for ROS Communication



Network

Select your network method

DHCP [✓ Static Address] Disabled network

✓ Network is connected

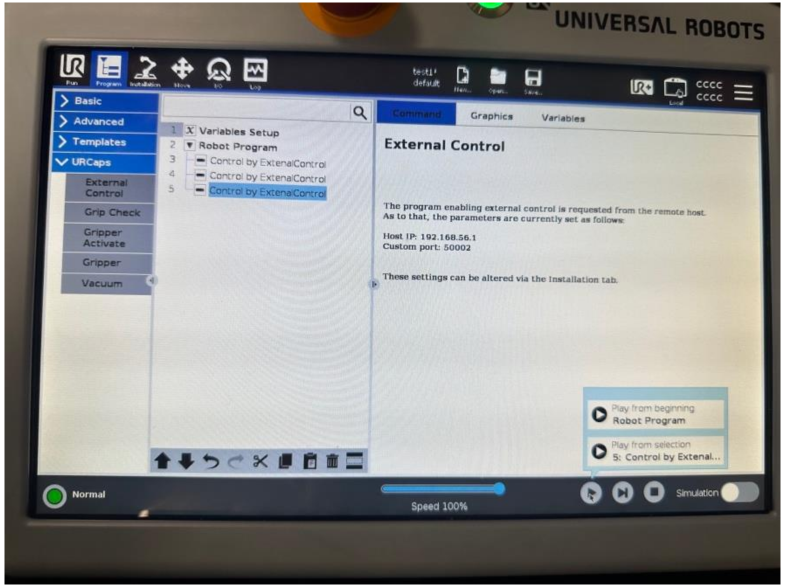
Network detailed settings:

IP address:	192.168.56.100
Subnet mask:	255.255.255.0
Default gateway:	0.0.0.0
Preferred DNS server:	0.0.0.0
Alternative DNS server:	0.0.0.0

Apply

•Set the UR controller to **Static Address** mode.
•Assign IP address (example shown: 192.168.56.100) that matches the ROS machine's subnet.
•Ensure correct subnet mask (e.g., 255.255.255.0).
•Apply settings and confirm the robot is "Connected" to the network.
•ROS PC and UR robot must be on the same network to enable External Control.

External Control URCap Setup



External Control

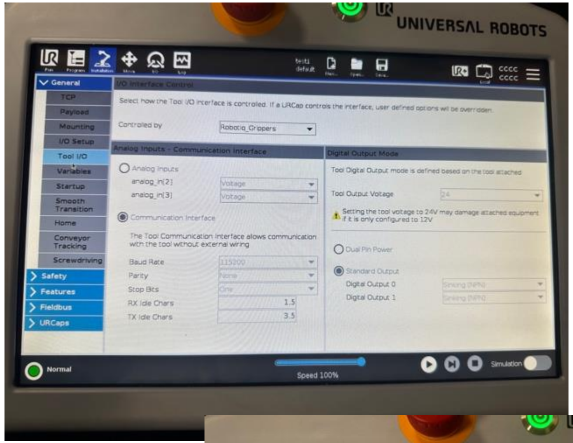
The program enabling external control is requested from the remote host.
As to that, the parameters are currently set as follows:

Host IP: 192.168.56.1
Custom port: 50002

These settings can be altered via the Installation tab.

•Under **Program → URCaps → External Control**, select *Control by ExternalControl*.
•This enables MoveIt to stream trajectories through the external_control node.
•Confirms host IP (ROS PC) and Ethernet port for communication.
•Required for autonomous execution of chess pick-and-place motions.

Robot Tool I/O & Gripper Interface



I/O Interface Control

Select how the Tool I/O interface is controlled. If a URCap controls the interface, user defined options will be overridden.

Controlled by: Robotiq_Grippers

Analog Inputs - Communication Interface

( ) Analog Inputs
analog_in[2]: Voltage
analog_in[3]: Voltage

(●) Communication Interface
The Tool Communication Interface allows communication with the tool without external wiring

Baud Rate	115200
Parity	None
Stop Bits	One
RX Idle Chars	1.5
TX Idle Chars	3.5

Digital Output Mode

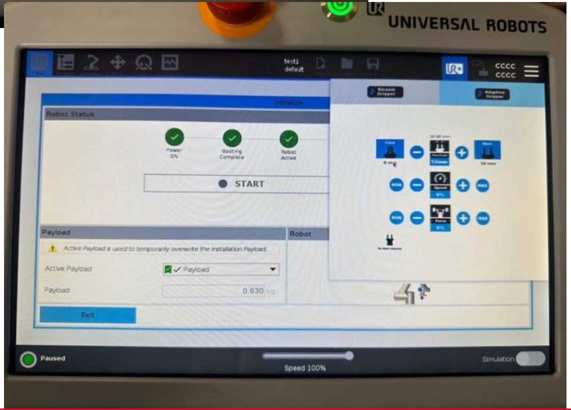
Tool Digital Output mode is defined based on the tool attached

Tool Output Voltage: 24

Setting the tool voltage to 24V may damage attached equipment if it is only configured to 12V

( ) Dual Pin Power
(●) Standard Output
Digital Output 0: Sinking (NPN)
Digital Output 1: Sinking (NPN)

Normal
Speed 100%
Simulation



Payload

Active Payload is used to temporarily overwrite the installation Payload.

Active Payload: ✓ Payload
Payload: 0.630 kg

Paused
Speed 100%
Simulation

•Configure the tool communication interface (e.g., RS485, voltage settings).
•Set appropriate digital output mode for gripper actuation.
•Tool output voltage must match the attached hardware to avoid damage.
•Even though our gripper malfunctioned, this configuration is required for future testing.
•Ensures consistent tool-state reporting for robotics pipeline.

Challenges

ROS Build & Dependency Conflicts

Initial build issues caused by conflicting ROS dependencies

Solution: Removed MoveIt and rebuilt pipeline without it

Gripper Hardware Malfunction

Gripper unreliable during testing, manual operation needed

Solution: Manual gripper fallback implemented

System & Network Instability

Device freezes, WebSocket drops, port/IP conflicts

Solution: Restarted device, added port management and auto-reconnect

Personal Device Setup Limitations

Tried migrating system to personal device for connections

Challenge: Time constraints prevented full environment setup

Limited Lab Access

One-week lab unavailability reduced physical testing

Solution: Focused on software integration, documentation, simulation

Demo 3 Individual Contributions

- **Abhay**
  - End-to-end vision-robot-webapp integration with IK support, waypoint management, WebSocket bridge, enhanced web interface, critical bug fixes, and SD07 website updates and Created the Hand-Off Document
- **Umesh**
  - .
- **Neha**
  - For Demo 3, I Refactored vision subsystem into standalone module by resolving cross-module dependencies and enabling autonomous operation without ROS or robot control
- **Logan**
  - Ensured waypoints for pick and place are accurate for every type of chess piece on every square and Edited and Uploaded Short DEMO video in the website

About Us

Abhay Prasanna Rao
Computer Science, DS & AI

abhay14@iastate.edu
Abhayprasannarao.com

Umesh Sai Teja Poola
Computer Science, DS & AI

upoola@iastate.edu
Umeshsaitejapoola.com

Neha Tirunagiri
Computer Science, DS & AI

neha2004@iastate.edu
nehatirunagiri.com

Logan Becker
CS, DS, Statistics

lbecker2@iastate.edu

Live Demo