Project Joseph Hoane: A Robotic Chess Player
By: SD02

Demo #2: Vision System Implementation

Bringing computer vision to chess automation

Team members:
- Junhyung Shim
- Cal Hokanson-Fuchs
- Autrin Hakimi
- Nhat Anh Bui
- Thanh Mai

Client: Dr. Bowen Weng

Computer Science Department

Iowa State University

04/08/2025

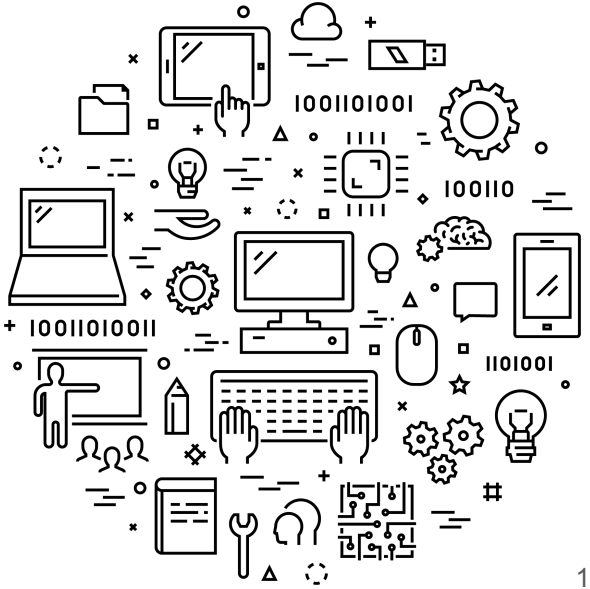
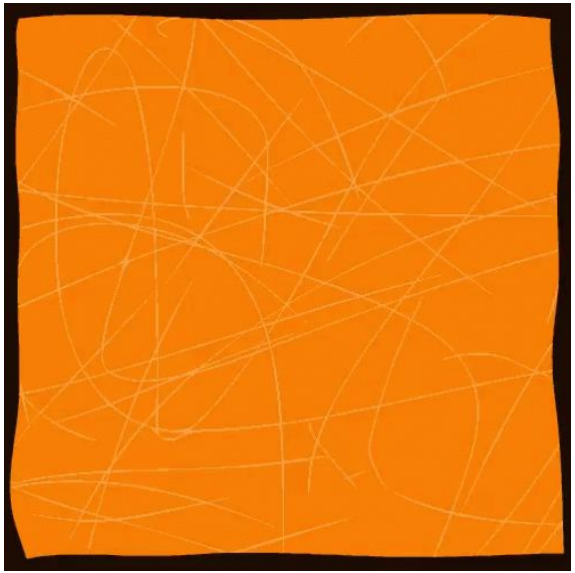


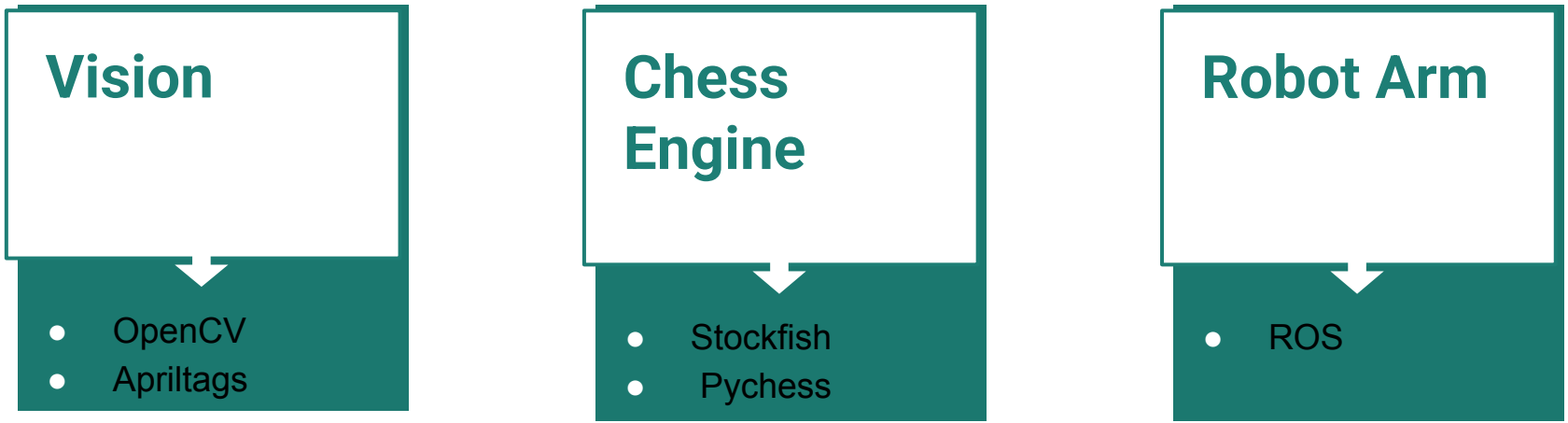
Table of Contents



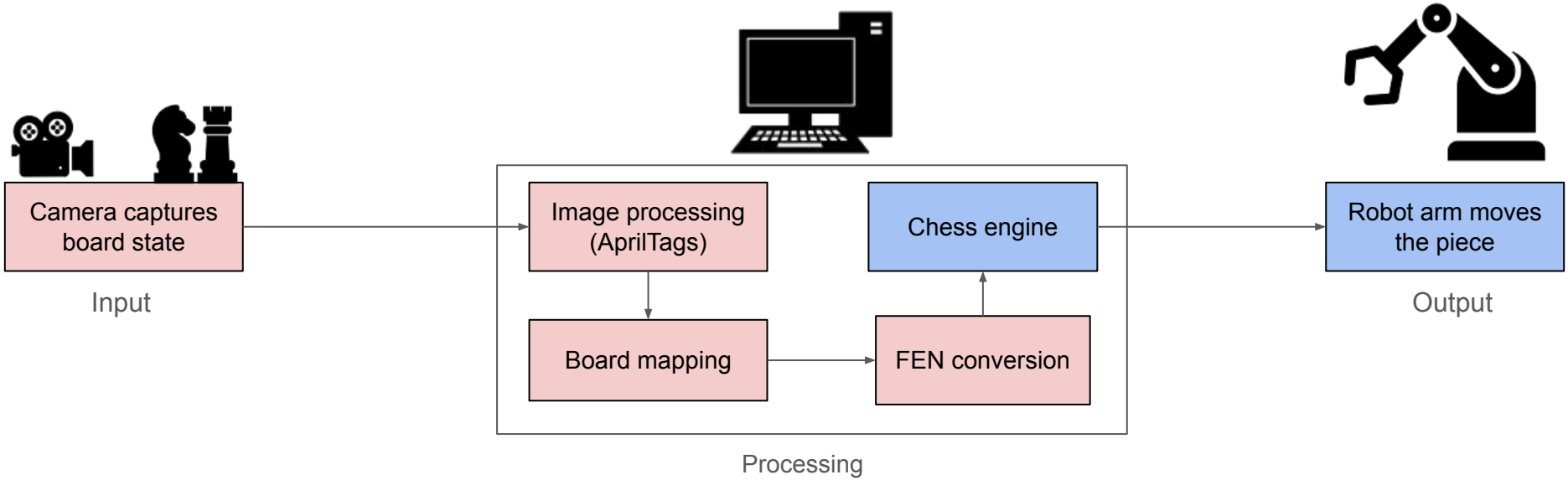
⟲ Recap

Project requirement:

- Develop a pipeline to allow a robotic arm to play chess against a human player



System Architecture



Vision system (highlighted in pink) is the focus of Demo #2.

📋 Demo #2 Requirements

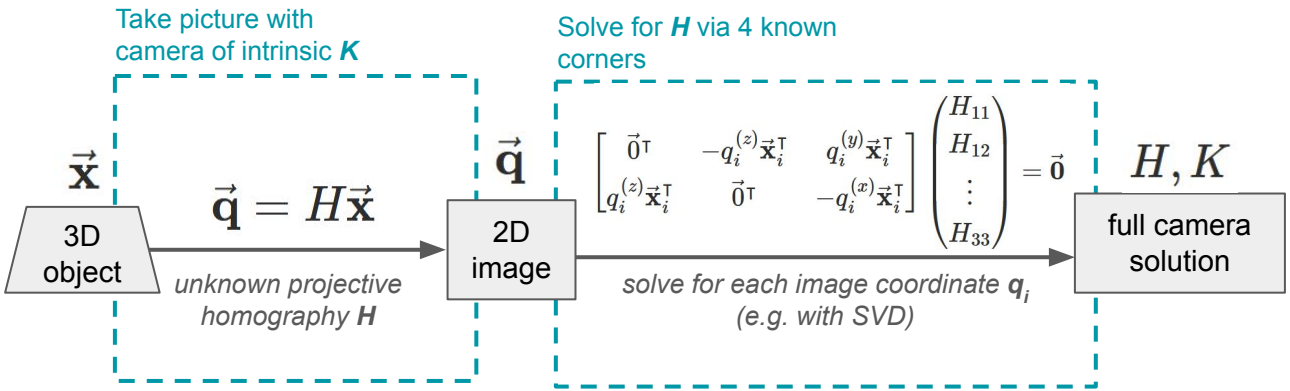
Board Detection (corner detection, transformation)	Piece Recognition (type identification, position tracking)
Chess Engine Integration (FEN conversion, game state)	Game Play (move tracking, validation) [Additional rules such as en-passant, promotions, and castling *are not required this demo.]*

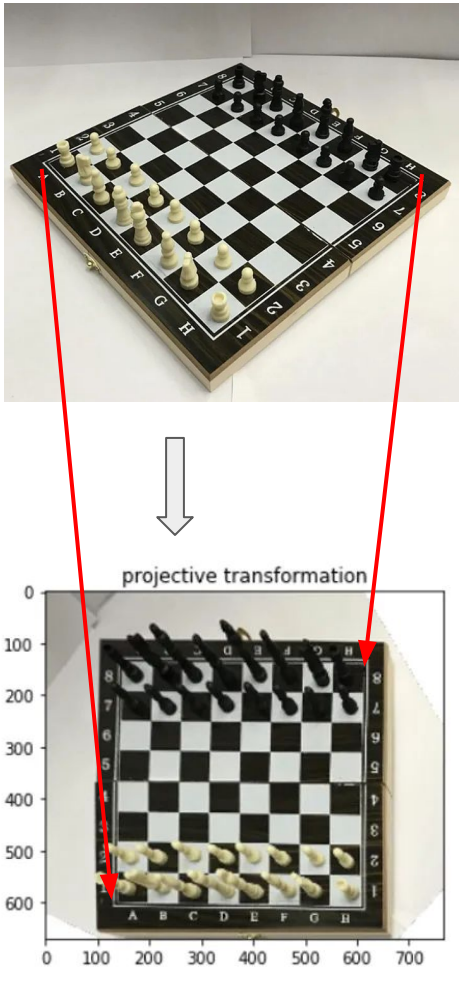
💼 Accomplishments

✅ 1. Used Apriltags + homography transformation to detect the chessboard
✅ 2. Developed mechanisms to determine chess piece positions and piece types using Apriltags and numpy
✅ 3. Developed a pipeline to convert the image into a Stockfish compatible input using PyChess
✅ 4. Developed a simple game playing mechanism to allow Interaction between a player and the engine

Homography Transformation

- A projective mapping between two planes
- Change a perspective image into a “bird’s eye view”.
- Map out the 4 corners of the source image and the final image, constructs a transformation matrix to change the source image.
- Use libraries to subdivide the board into 8x8.



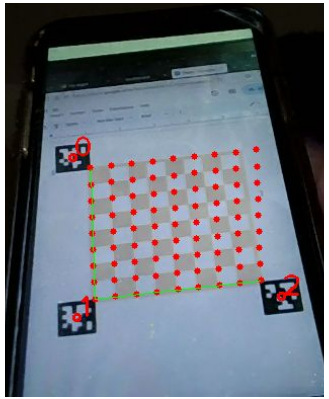


Implementation - Detecting the Chessboard

Before:

- Using 3 reference points:

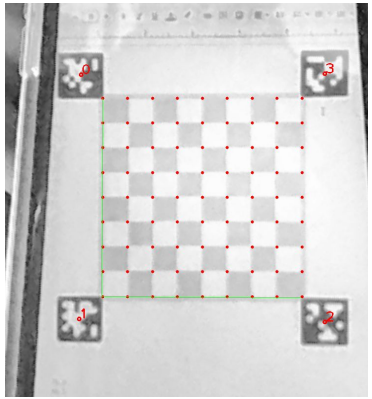
  Accurate detection of the board



After:

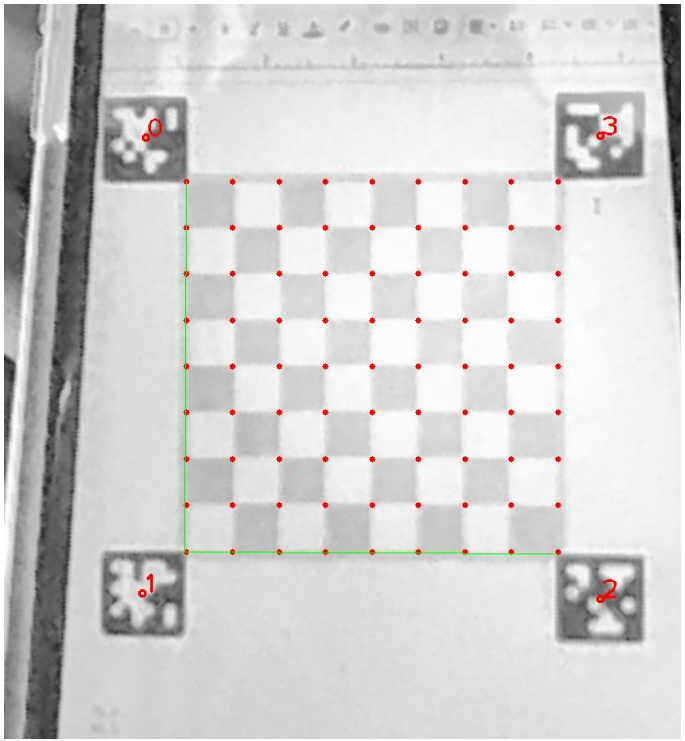
- Using 4 reference points + applying the homography transformation:

  Accurate detection of the board



Implementation - Detecting the Chessboard

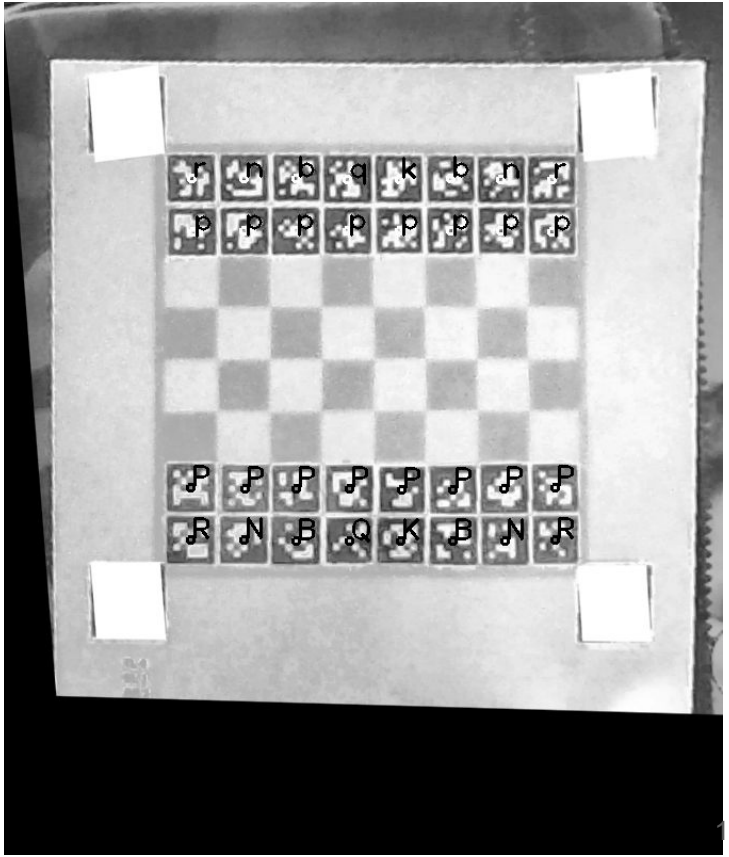
- We can measure the relative width and length of the board using tags
- Apply np.meshgrid to make the grid
- 3 unique vertices per square
- cells[8][8] contain x and y boundaries of each cell e.g.
    - cells[0][0].TL = (0,300)
    - cells[0][0].BL = (0,0)
    - cells[0][0].BR = (300,0)



Implementation - Detecting the Pieces

Used April tags again for identifying the pieces

- Used 2 different family of tags
  - 36h11 (id: 4 ~ 19) for B
  - 25h9 (id: 0 ~ 15) for W
- Map each id to their designated piece e.g. 25h9-id0 as white side pawn



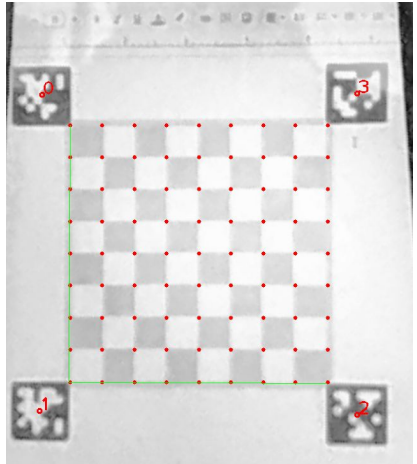
Implementation - Identifying Piece Locations

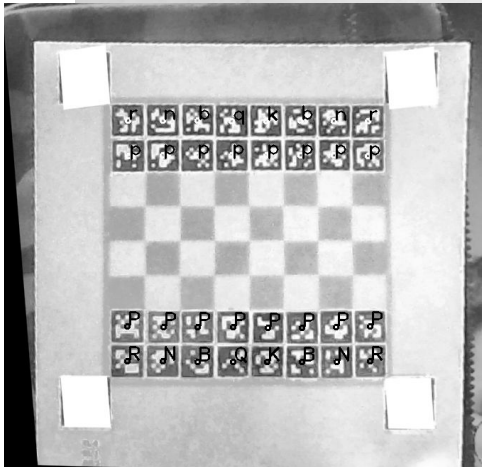
- (x,y) position of the center of a tag is given by the detector
- For each tag, find the square where the square's boundary encloses the center vertex of the tag.

For p in pieces

For c in cells

if p.tag.center is within c, report p in cell c





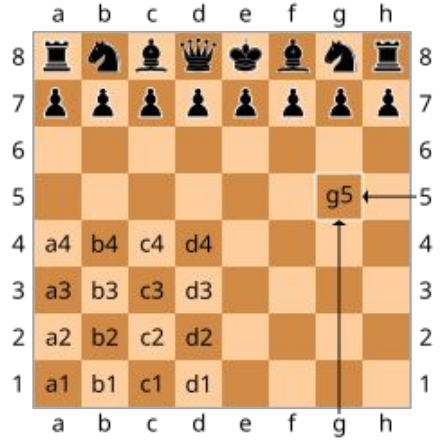
Integration with Chess Engine

- Directly converting the board state into FEN string is time consuming and error prone.

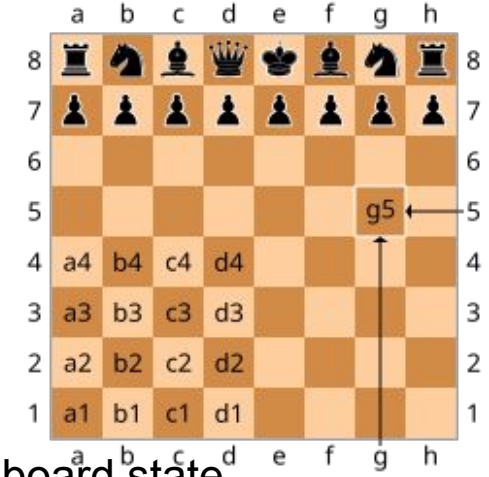
  → Pychess can track the board states by tracking each player's moves
  → Pychess can return the FEN string of the current state of the board

- Pychess accepts algebraic notation
  i.e. a2a3 means move the piece at a2 to a3

  → We just need to convert a player's move into the corresponding algebraic notation



Integration with Chess Engine



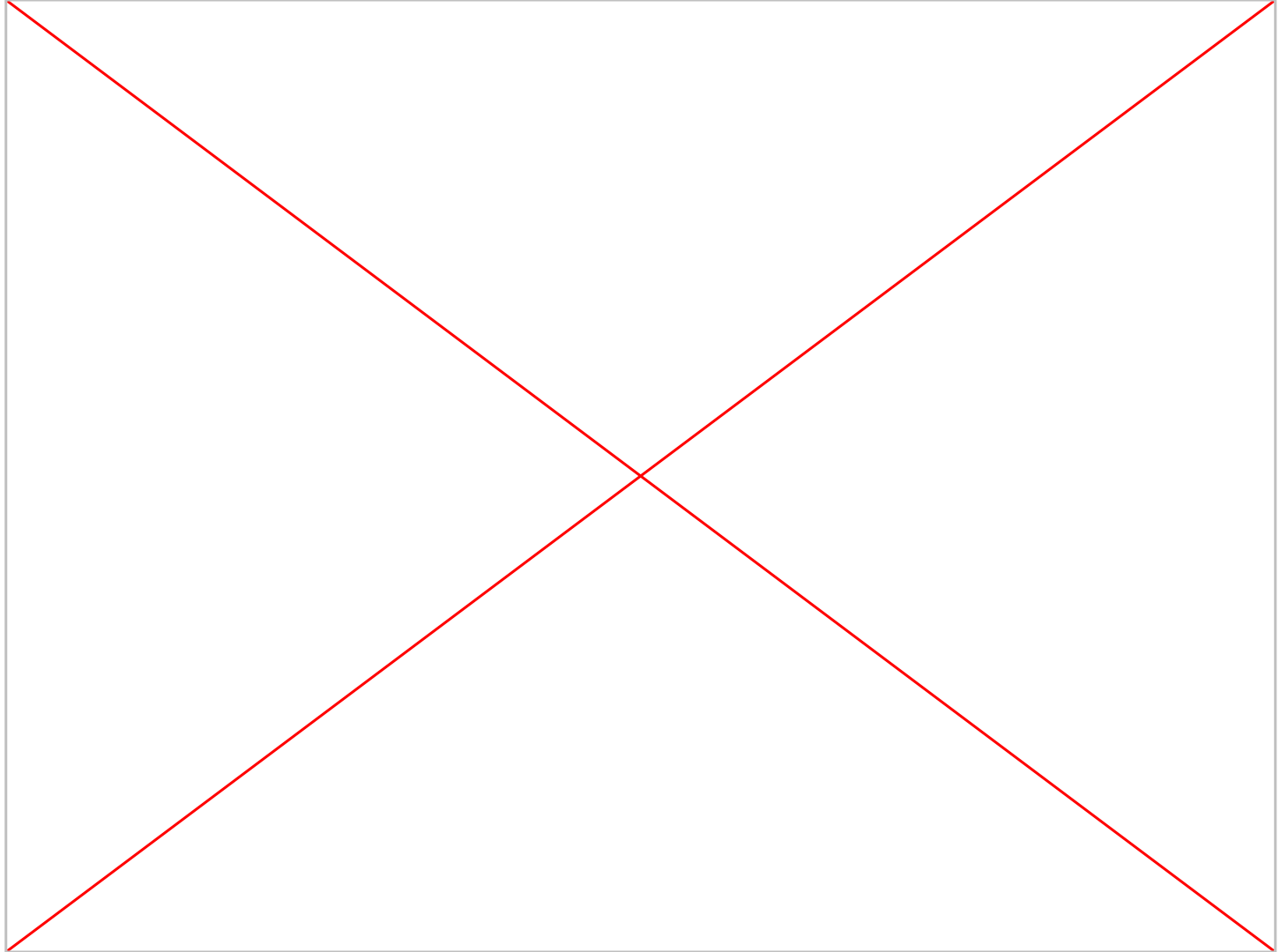
Player's turn:

1. Save the current board state
2. Pause the camera feed and let player make the move
3. Turn on the camera again and detect which piece moved
4. Generate the algebraic notation e.g. "a2a3"
5. Feed it into Pychess, and fetch the FEN string of the current board state

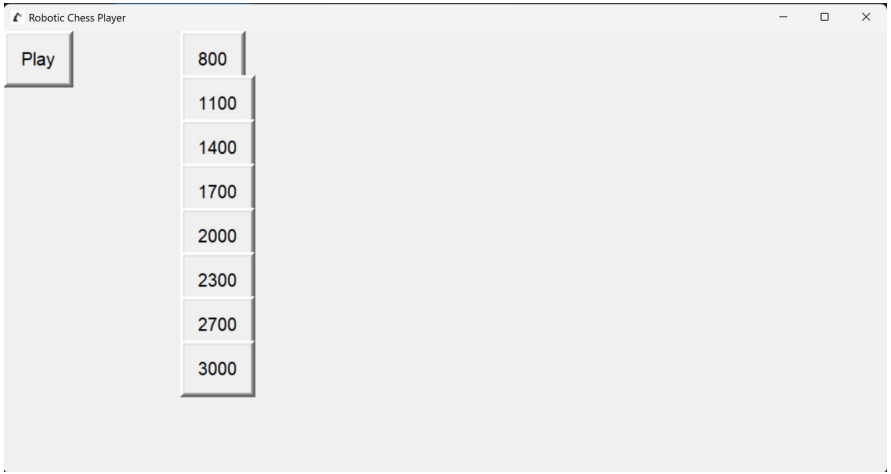
AI's turn:

1. Stockfish returns its desired move in algebraic notation
2. We output which piece (identified by tag-id) needs to move and its destination
3. Once the piece enters the target square's boundary, we report the move to Pychess
4. Pychess records the move, and then the opponent takes their turn.

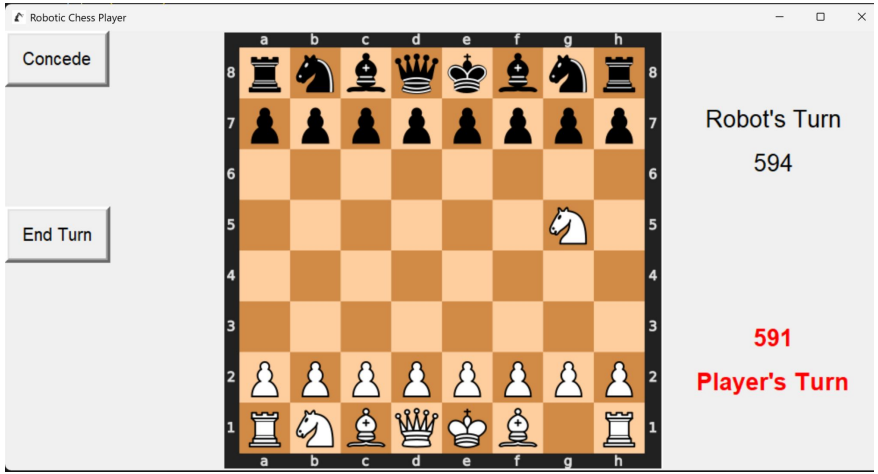
Demo - Video



Frontend

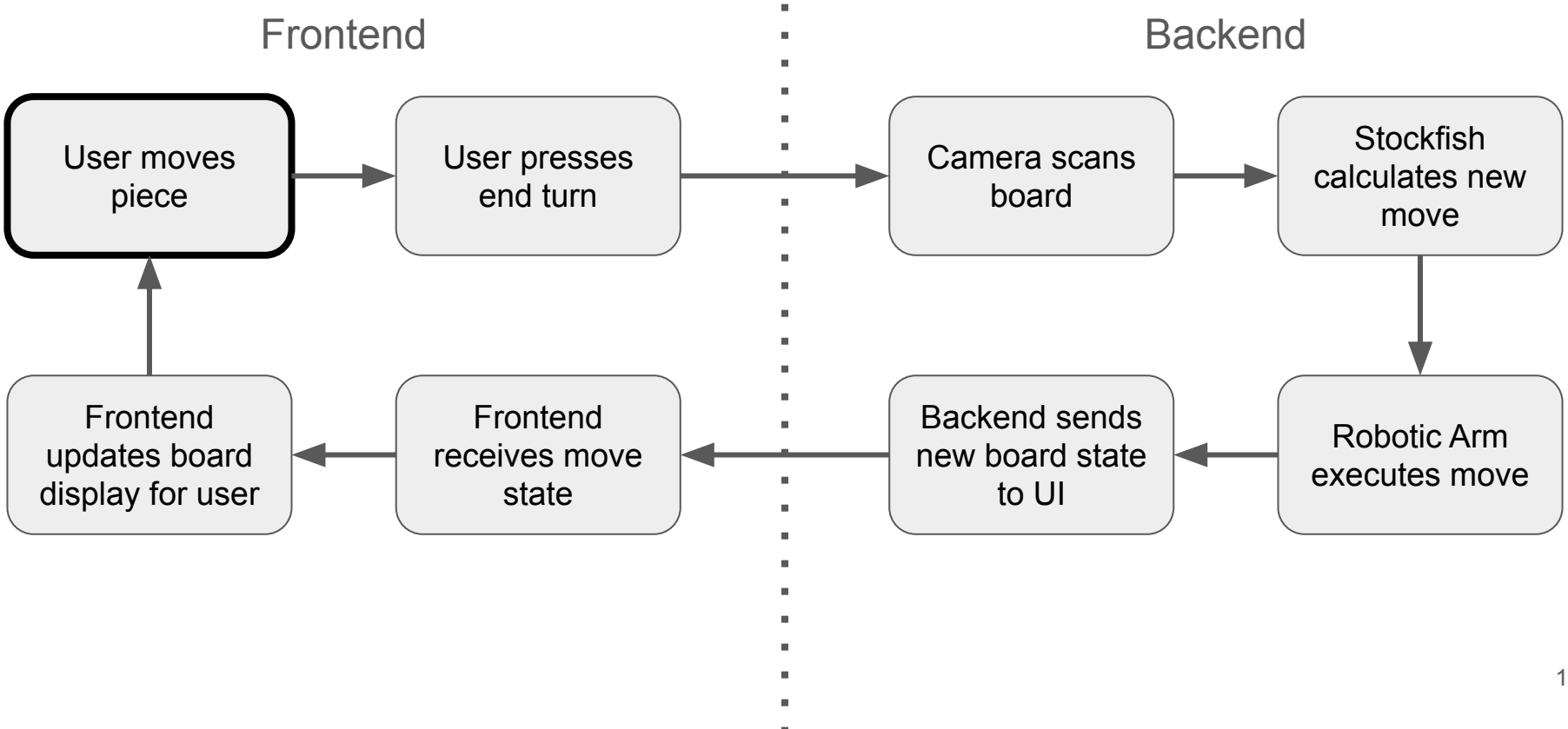


Home Page



In-Game

Game Loop

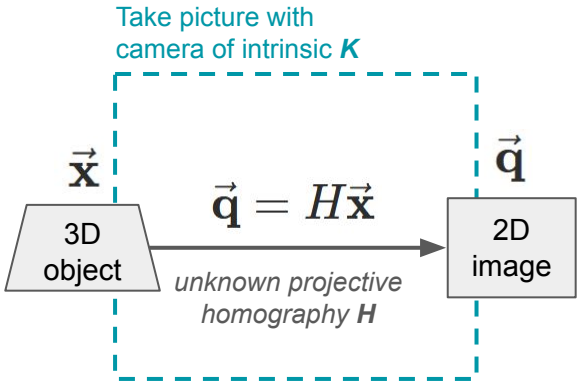

Frontend
Backend
User moves piece
User presses end turn
Camera scans board
Stockfish calculates new move
Robotic Arm executes move
Backend sends new board state to UI
Frontend receives move state
Frontend updates board display for user

Robotics

- Goal: translate a chess move into a physical robot action
- Step 1: Reconstruct chessboard topology in 3D space
- Step 2: Plan required actions and trajectories to play a move
- Step 3: Control physical robot arm through ROS *(in progress)*

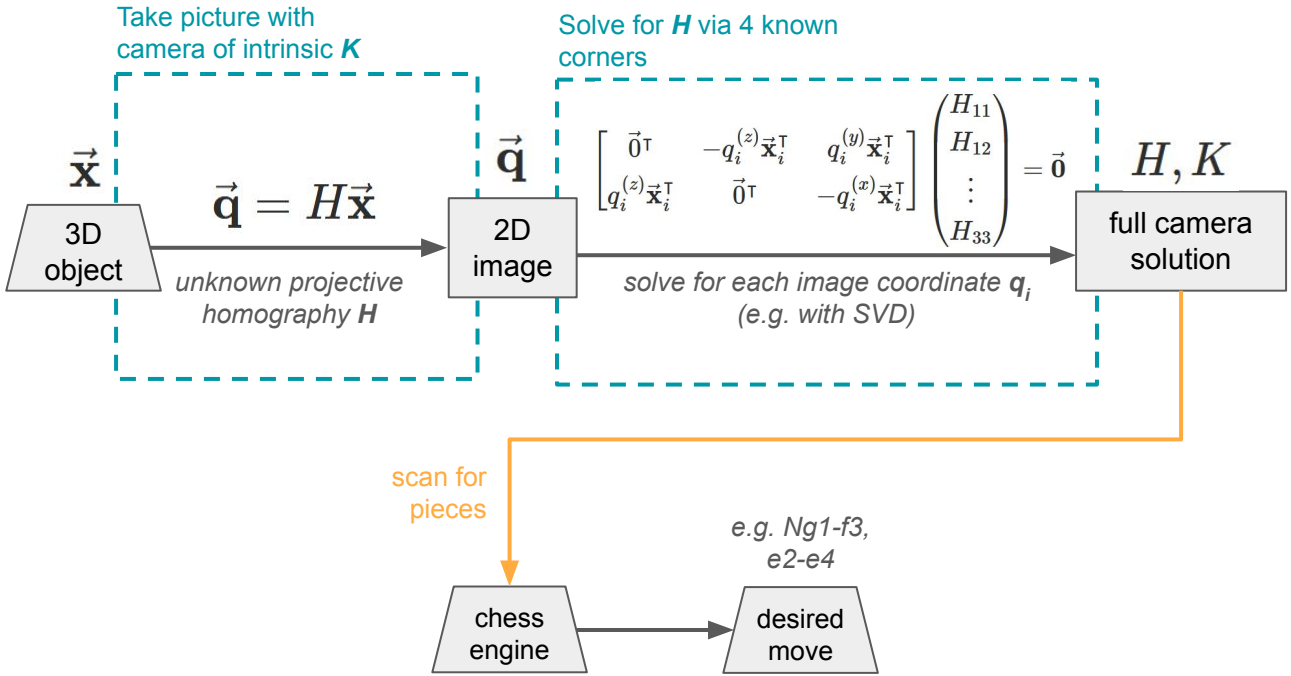
Robotics - 3D reconstruction

- Recall that a homographic projection H was computed from 4 points



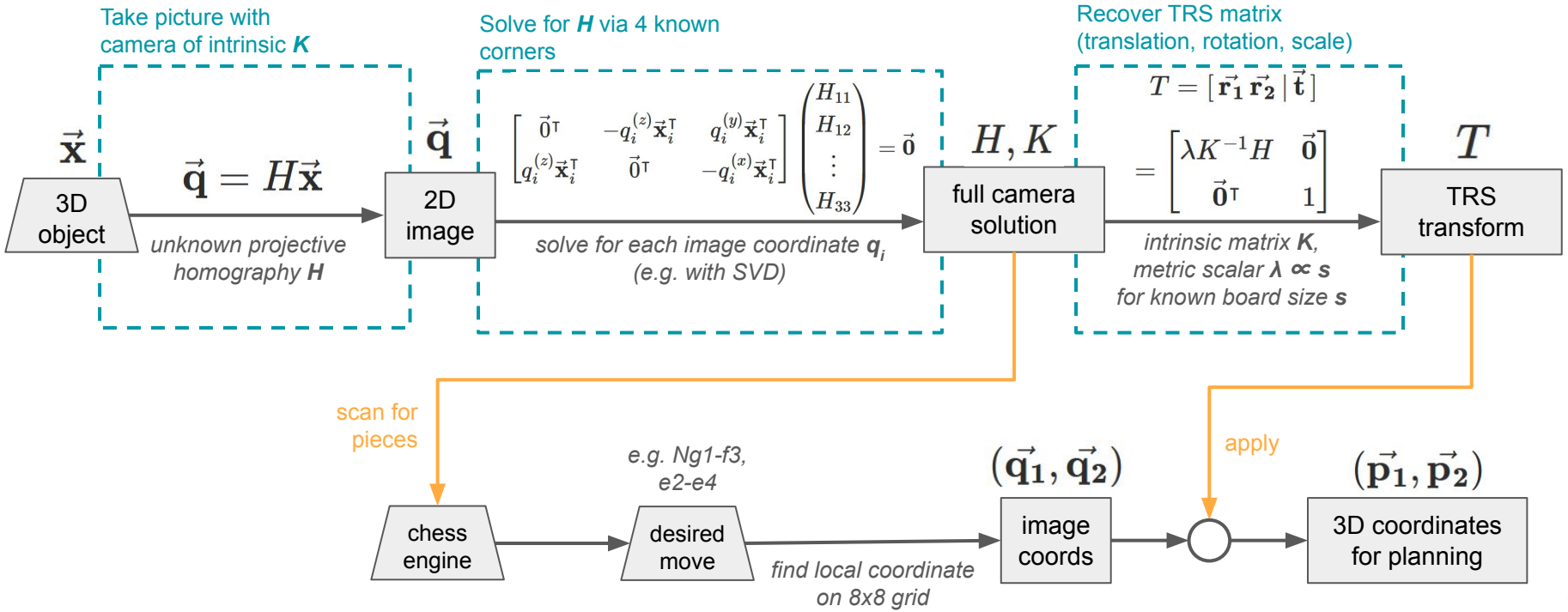
Robotics - 3D reconstruction

- Recall that a homographic projection H was computed from 4 points



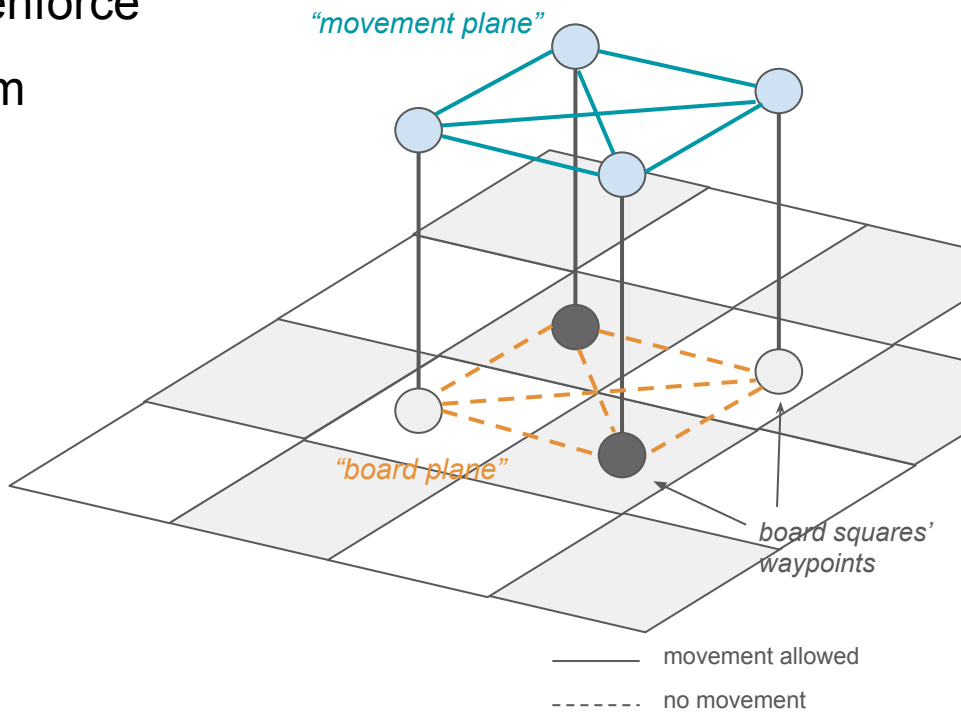
Robotics - 3D reconstruction

- Recall that a homographic projection H was computed from 4 points
- Use this to convert chess moves back to physical 3D movement



Robotics - Waypoint graph

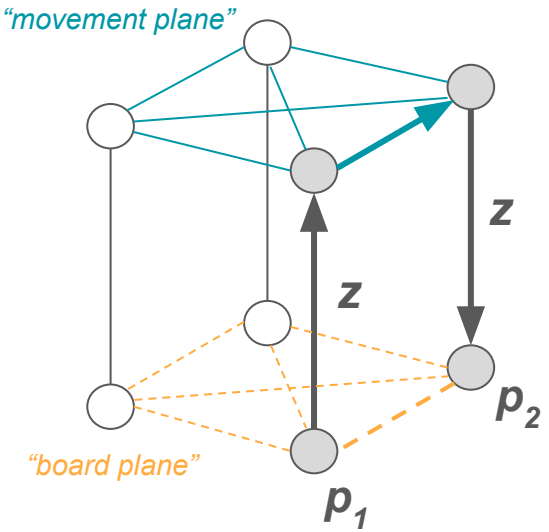
- Construct a graph of “waypoints” to enforce possible movements for the robot arm
  - Avoid contact with surface
  - Avoid contact with other pieces
- Allows planning trajectory arcs
- Special positions for:
  - Idling
  - Discarding taken pieces



Robotics - Movement planning

"**Move**" action chain to move piece from p_1 to empty target p_2:

1. Move to just above the piece $p_1 + z$
2. Descend to grabbing position p_1 and grab the piece
3. Ascend back up and move to above the target $p_2 + z$
4. Descend to target p_2 and drop the piece
5. Ascend back up

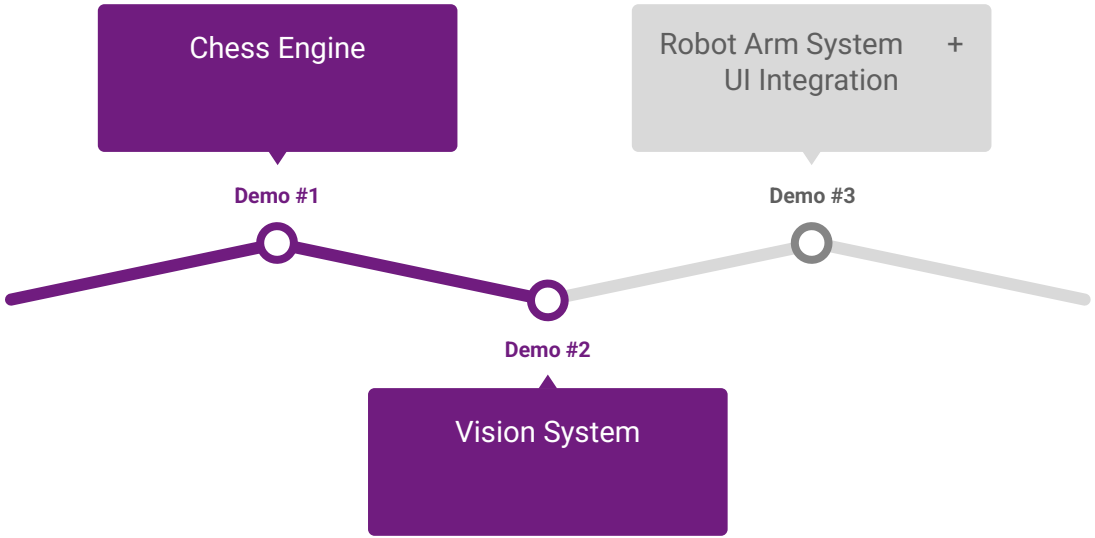


Notes:
- ***z** must be taller than the pieces' height*
- *board plane must just barely hover above the actual board*

Robotics - Special moves

- Piece A takes piece B at ***p***:
  - **Move** piece B to discard position outside the board
  - **Move** A to ***p***
- Castling:
  - **Move** king to G file (short) or C file (long)
  - **Move** rook to F file (short) or D file (long)
- Challenges: pawn promotion, bad placement by human player, etc.

Next Steps



- Robot Arm Integration
- Movement planning for chess pieces
- Safety protocols for human interaction
- Full game testing and validation
- Frontend-Backend Connectivity

Challenges

- Lighting variation affecting AprilTag detection
- Camera positioning and calibration
- Processing speed optimization
- Distinguishing similar pieces
- Connecting frontend to backend


Thank
You