



# REAL-TIME CHESS GAME ANALYZER

*Serden Sait Eranil  
Veysel Colak  
Firdevs Su Aydin*



# Outline

- Motivation
- Project Steps & Work Distribution
- Project Pipeline (ROS Nodes & Topics)
- Deeper Look Into Project Steps
- Accuracy and Delay Results
- Demo
- Future Scope

# Motivation

- Automate recording of the moves
- Instead understand the moves with camera and save them

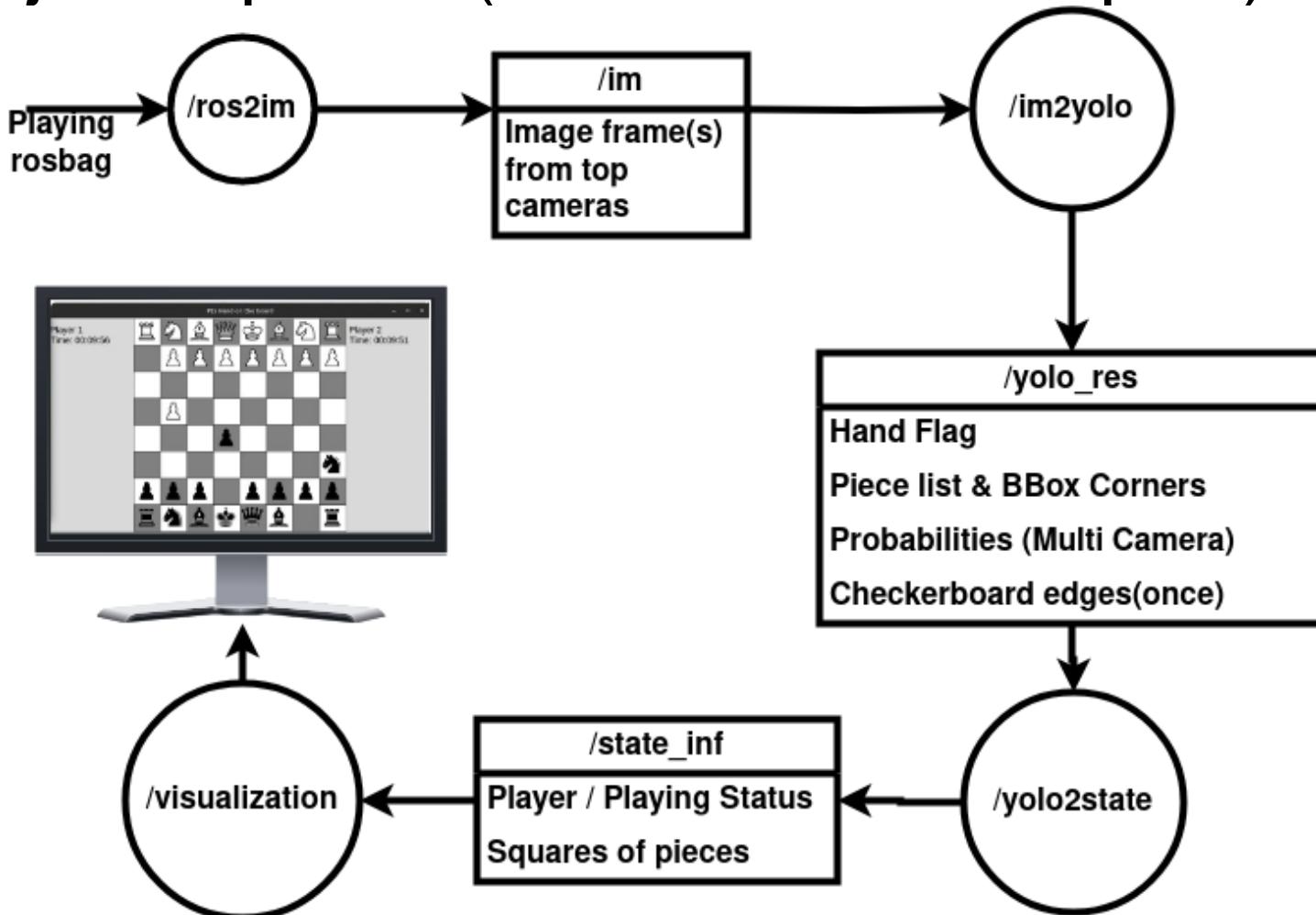




# Project Steps & Work Distribution

Data Collection & Annotation	YOLO Model Selection	Piece, Checkerboard and Hand Detection
Serden Su Veysel	Su Serden	Su Serden
Game State Management	Visualization	ROS Pipeline & Workspace
Veysel	Serden	Veysel

# Project Pipeline (ROS Nodes & Topics)

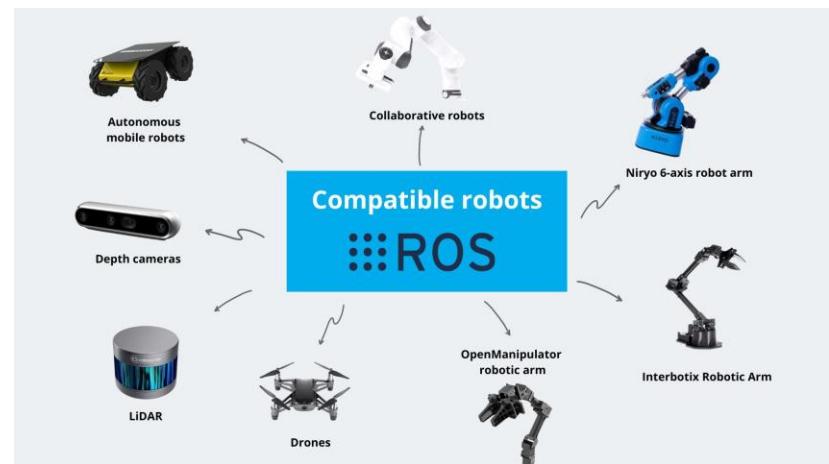




# **Deeper Look Into Project Steps**

# ROS

- Robot Operating System is...
  - a robotics workframe.
  - a library for C and Python.
  - a middleware between distributed system.
- "Nodes", "Topics", "Services", "Actions" ...





# What does ROS bring?

- The "communicating nodes" structure
  
- In robotics domain also...

# But there are two of them!

- ROS has two versions: ROS & ROS2.
  - No Masters
  - Message QoS profiling
  
- Due to lab computers and bag files,  
we have moved our project to ROS1

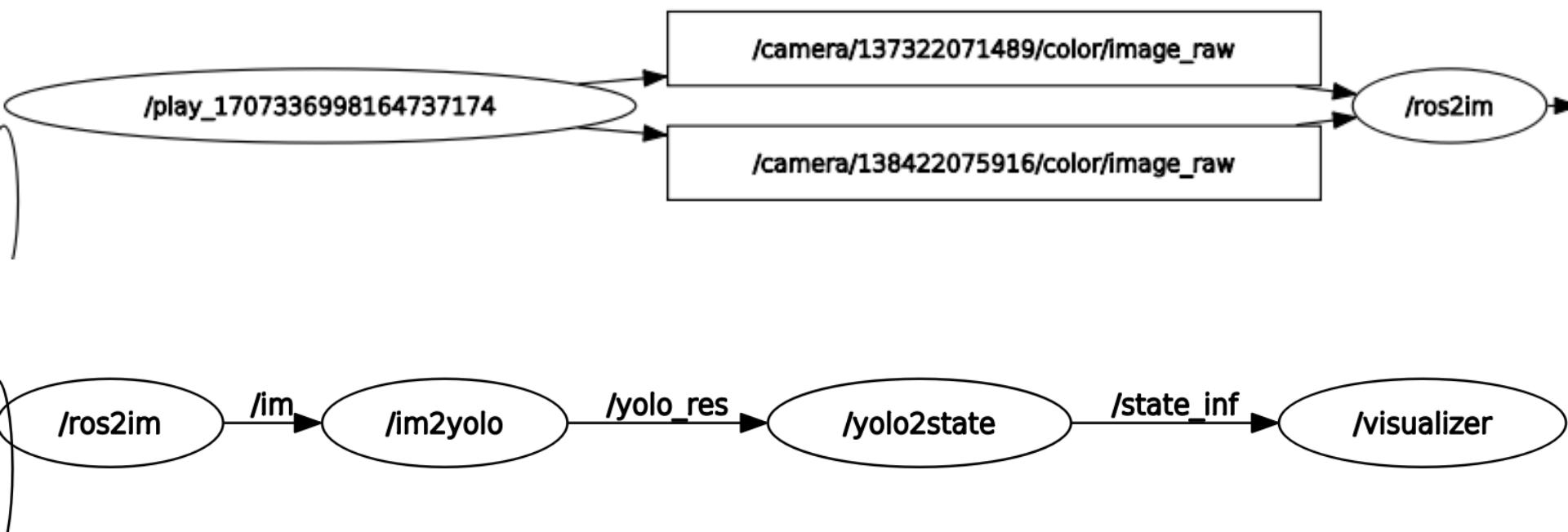


The logo for ROS 2 consists of a dark blue vertical stack of six dots followed by the text "ROS 2" in a large, bold, dark blue sans-serif font.



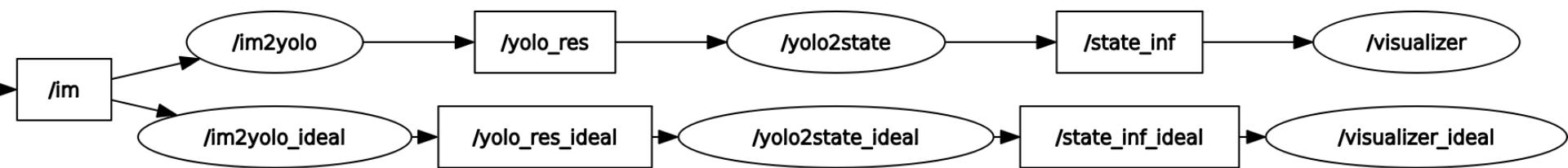
The logo for ROS consists of a dark blue vertical stack of six dots followed by the text "ROS" in a large, bold, dark blue sans-serif font.

# ROS Implementation (with Camera Node)



# ROS Implementation (with Test Node)

- Utilize the ROS node2node communication
- On parallel use a "perfectly arranged determiner"
  - Use the branch for each step after "/im" topic



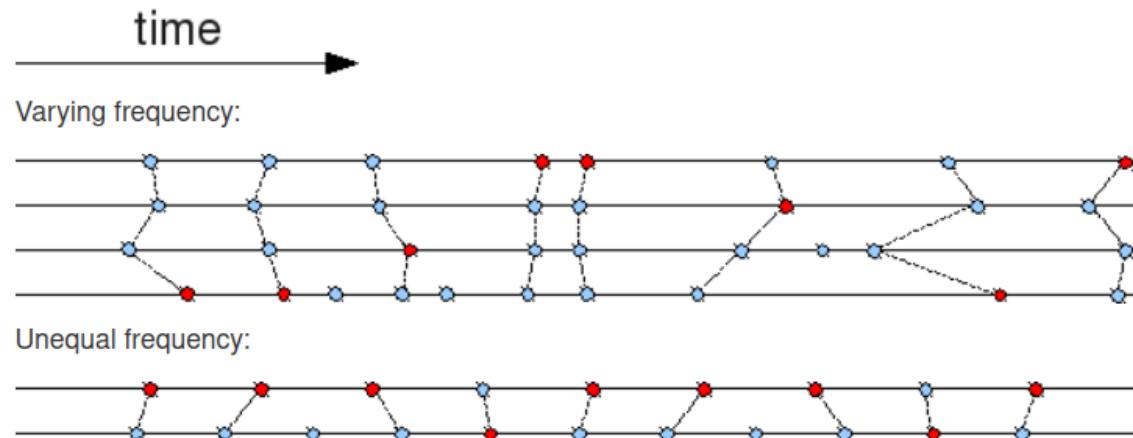
# ROS Reading Camera (/ros2im)

- Other nodes subscribe 1 topic
  - But /ros2im subscribes 2 topics
  - They should be combined.
  
- Approximate Time Synchroniser



# Time Synchronization in ROS

- TimeSynchronizer:
  - Perfect but Unimplementable
- ApproximateTimeSynchronizer:
  - Frame loss possible but works

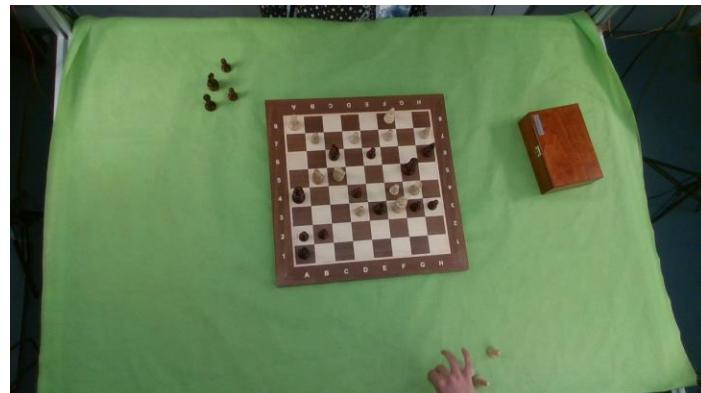




# **DATA ANNOTATION TRAINING AND TEST**

# Data Collection and Annotation

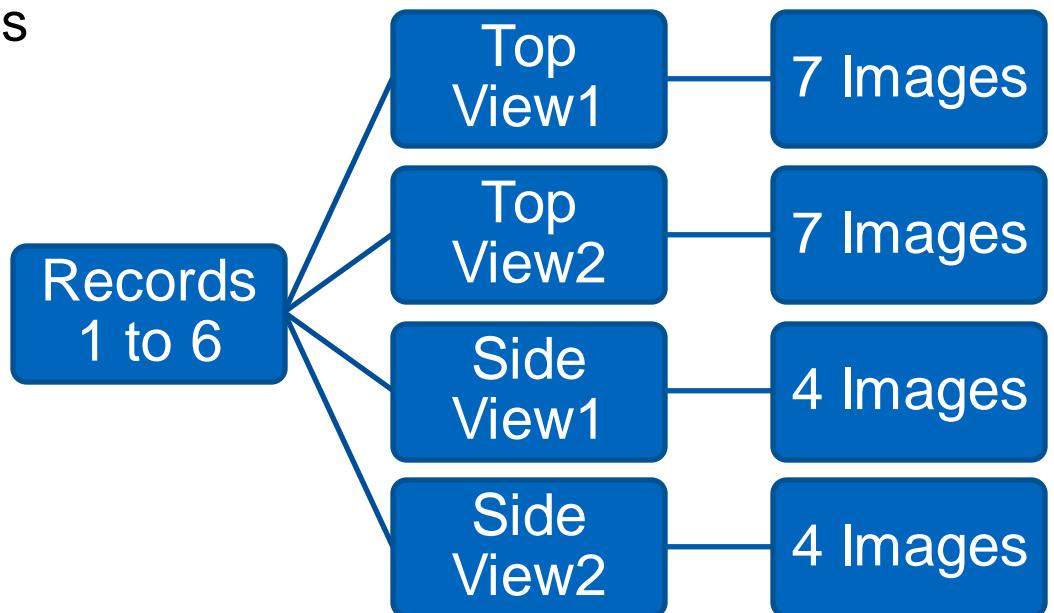
- 8 Recordings
- 4 Cameras: 2 top views, 2 side views



# Data Collection and Annotation

Training Dataset:

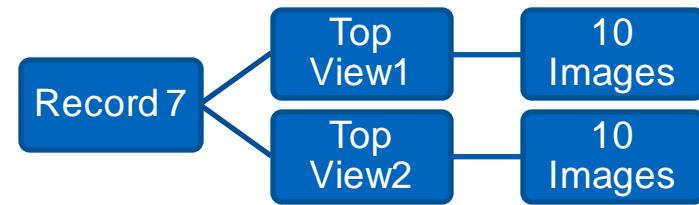
- 6 recordings, 4 cameras
- Total of 132 training images



# Data Collection and Annotation

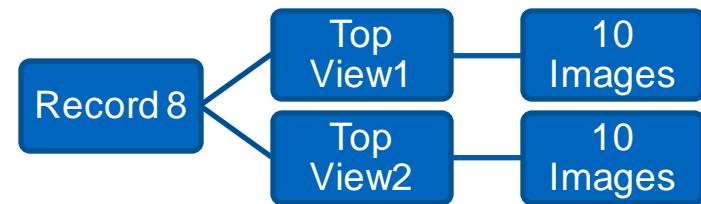
## Validation Dataset:

- 1 recording, 2 top views
- Total of 20 validation images



## Test Dataset:

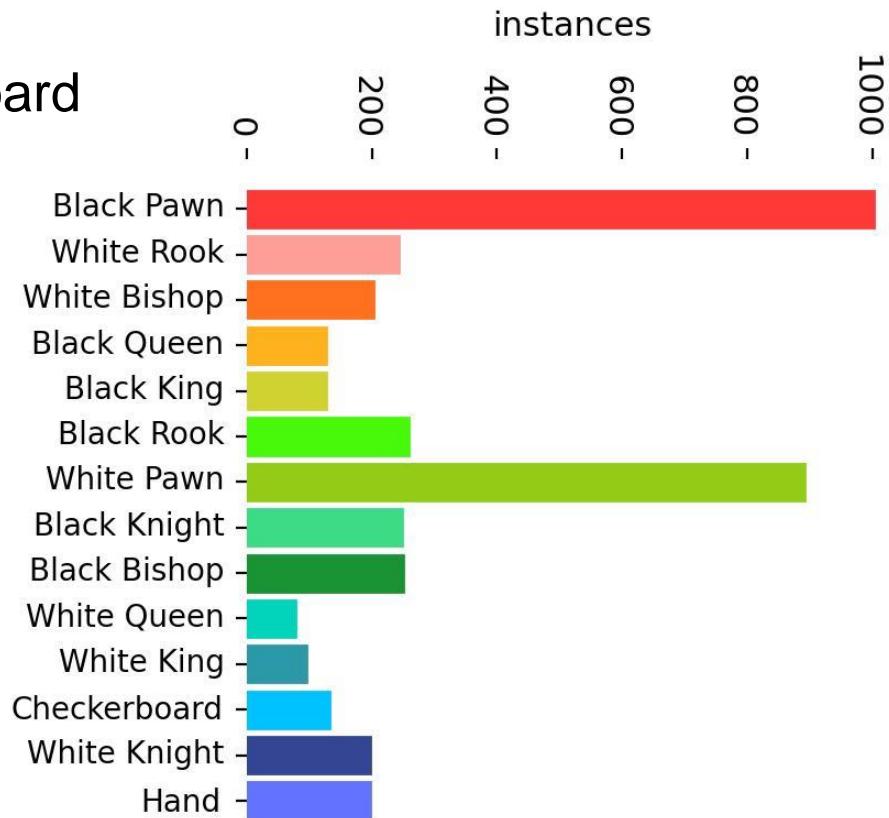
- 1 recording, 2 top views
- Total of 20 test images



# Data Collection and Annotation

## Data Distribution:

- 12 piece + hand + checkerboard
- Not evenly distributed



# CVAT: Computer Vision Annotation Tool

- Easy-to-use, open-source, collaborative web-based tool
- Allows the use of state-of-the-art segmentation methods (e.g., Segment Anything, HRNET)

### Pphau Chess Annotation

Project #98141 created by eranil17 on February 1st 2024

Assigned to

**Project description**

[Edit](#)

**Issue Tracker **

[Raw](#) [Constructor](#)

Add label  [Setup skeleton !\[\]\(0fb9f80ff2b6093ef9718f74d0122401\_img.jpg\) \[From model !\\[\\]\\(ec0cb51eec8f44f45e77beb9e5ec4dc9\\_img.jpg\\)\]\(#\)](#)

[Black Pawn !\[\]\(376cb9d99f8a3226845d540524970bec\_img.jpg\) !\[\]\(1e27f603637647b99bad02aba52370d6\_img.jpg\)](#) [White Rook !\[\]\(4108625183666eae4686a1632fe22eae\_img.jpg\) !\[\]\(7b92f7f21ca75aa09df718f58c5ed20f\_img.jpg\)](#) [White Bishop !\[\]\(902256d3ed8b1e74f5fb1394e7f0aa96\_img.jpg\) !\[\]\(3f37989d3cfb13fc6b2bc97a540e9ac0\_img.jpg\)](#)

[Black Queen !\[\]\(6fd08ad500cbdfcd0f44805a13205ca0\_img.jpg\) !\[\]\(3b2f6a9e665559f14c4424bccfb60e9f\_img.jpg\)](#) [Black King !\[\]\(e30050383c9bd09fa7f949458f6292b6\_img.jpg\) !\[\]\(e46511846719e5bc25e5abafcdff2b79\_img.jpg\)](#) [Black Rook !\[\]\(03f65dd3c544336fb2fb0137704ad9a0\_img.jpg\) !\[\]\(c648eacfaca1770355fa05c0f57bf097\_img.jpg\)](#) [White Pawn !\[\]\(e20f4ee289e55db7636896c2a293381a\_img.jpg\) !\[\]\(994468bff3f399c0783675df121de8ec\_img.jpg\)](#) [Black Knight !\[\]\(b8594b6f76422fae2b0f117510f6b5f0\_img.jpg\) !\[\]\(dffc63bb5a211eb1bbcaf9a9d01f3057\_img.jpg\)](#)

[Black Bishop !\[\]\(8b09d1cf4590309032c55797f6ad9502\_img.jpg\) !\[\]\(997a8404ae9f6a05d9a027b02eae970d\_img.jpg\)](#) [White Queen !\[\]\(bcc422d06ab91b8a48b3e8c4d29d7148\_img.jpg\) !\[\]\(831766b748b0efef5c81204ead8b194b\_img.jpg\)](#) [White King !\[\]\(3a91ee9940a27bdc04302d21f6d161a9\_img.jpg\) !\[\]\(dc6fc1e9973887e75dc7b5f5105905de\_img.jpg\)](#) [Checkerboard !\[\]\(22885662f65f4c0ac7dc30cc1f26dc33\_img.jpg\) !\[\]\(c129e5a52fc9a6ee0e87fa7a84027d63\_img.jpg\)](#) [White Knight !\[\]\(d86ba3393532b8f8b5348bb5a39af2e0\_img.jpg\) !\[\]\(c4d8219fde6d6a6d85d3fe1c6f3d5dd0\_img.jpg\)](#)

[Hand !\[\]\(7e3858db09ddfad759171b61b52d071b\_img.jpg\) !\[\]\(289680677fca5d06f006abff988b7b5a\_img.jpg\)](#)

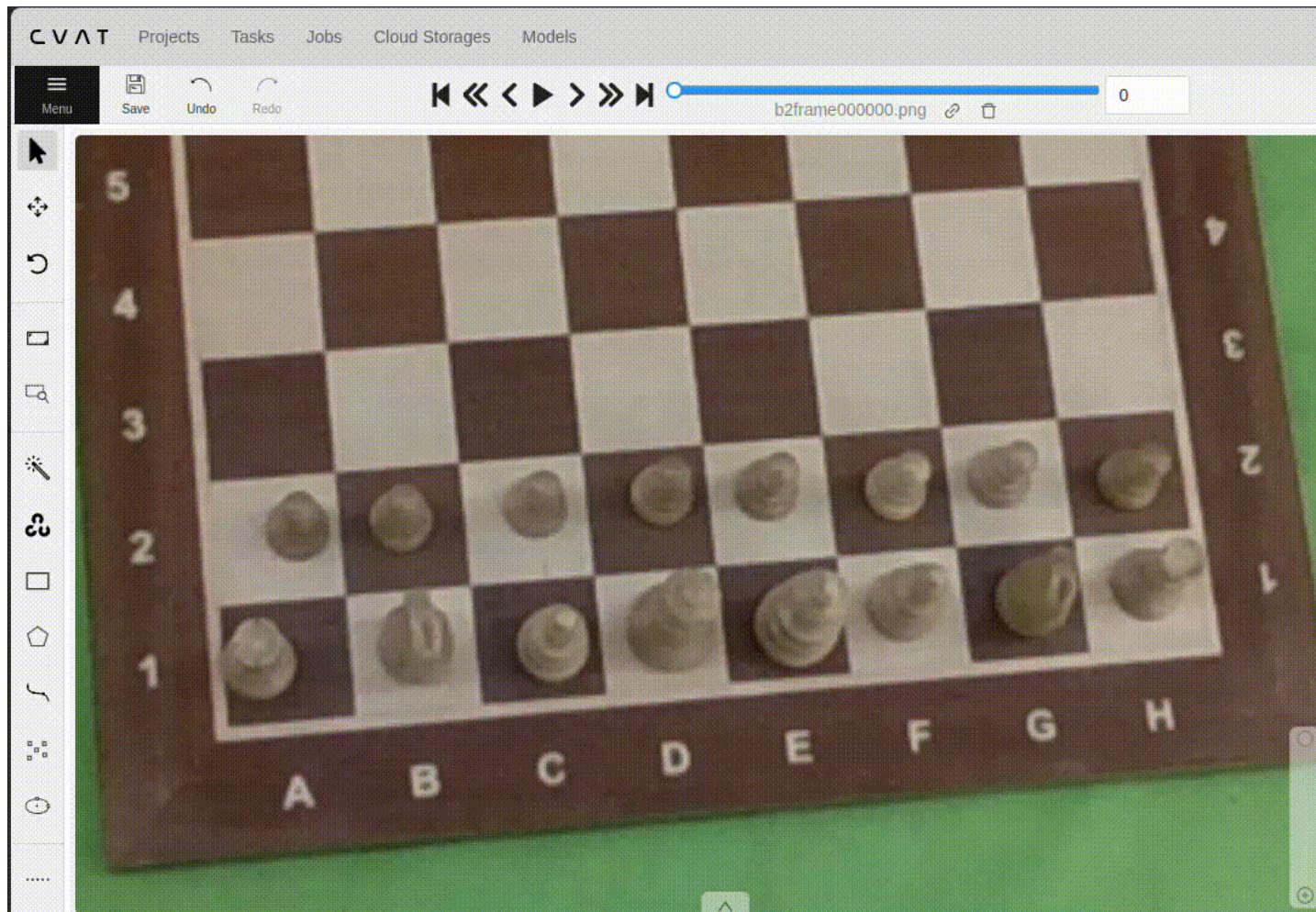


# Why didn't we use MIVOS?

- Poor quality of the pieces
- Insufficient support of instances
- Fast hand movements
- Too chaotic piece distribution

**Solution: Through use of CVAT, add some manual input**

# How did we do the annotation in CVAT: GIF



# Final annotation outcome per frame





# YOLO MODELS

# YOLO Model Selection

- 4 sizes of YOLO v8 model:
  - Nano
  - Small
  - Medium
  - Large



Model	size (pixels)	mAP <sub>val</sub> 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
<a href="#">YOLOv8n</a>	640	37.3	80.4	0.99	3.2	8.7
<a href="#">YOLOv8s</a>	640	44.9	128.4	1.20	11.2	28.6
<a href="#">YOLOv8m</a>	640	50.2	234.7	1.83	25.9	78.9
<a href="#">YOLOv8l</a>	640	52.9	375.2	2.39	43.7	165.2

Figure: Parameters of YOLO models. Adapted from <https://github.com/ultralytics/ultralytics>

# YOLO Model Selection

## Training:

- Avoid overfitting.

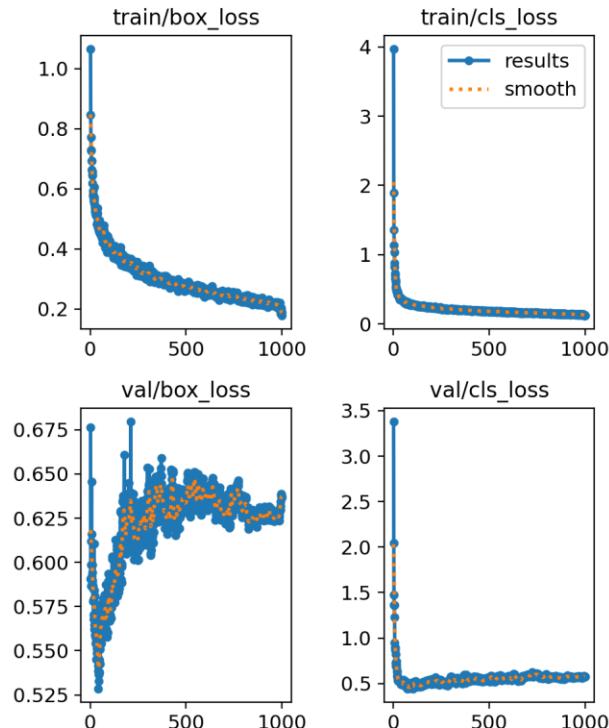


Figure1: Training  
curves for small model.

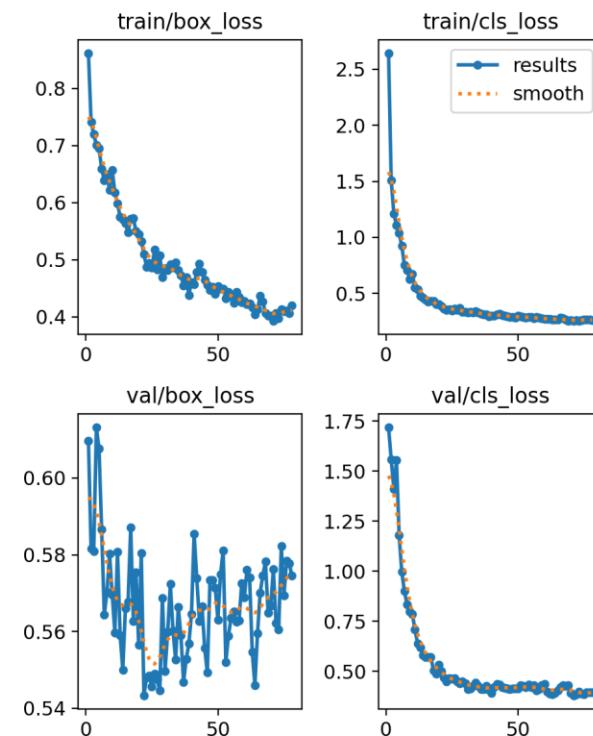


Figure2: Training curves for  
medium model.

# YOLO Model Selection

## Test: Confusion matrices

- Some pieces are confused with each other.



Figure 1: Confusion matrix for large model with test data

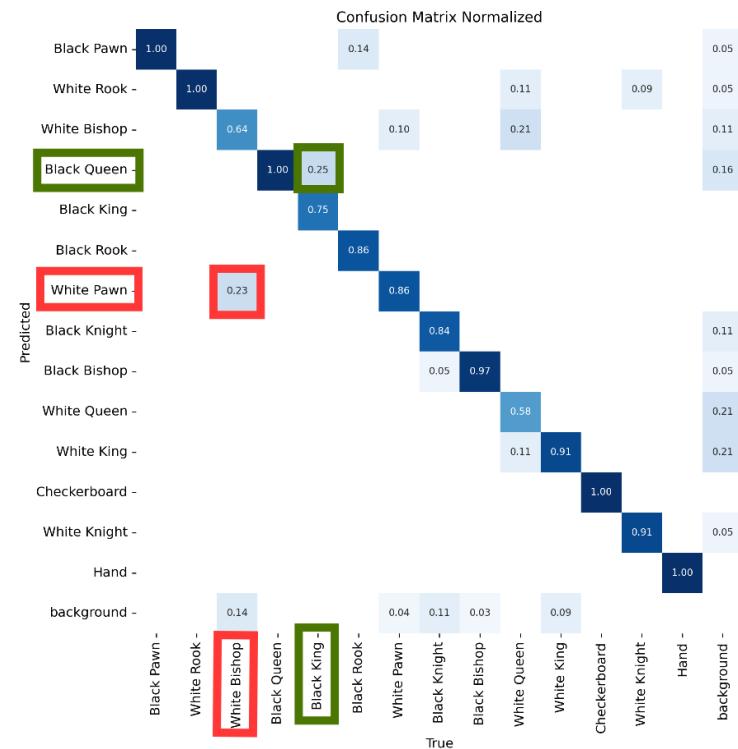


Figure 2: Confusion matrix for small model with test data

# YOLO Model Selection

Test: F1 Scores

- Kings and Queens are harder to detect.

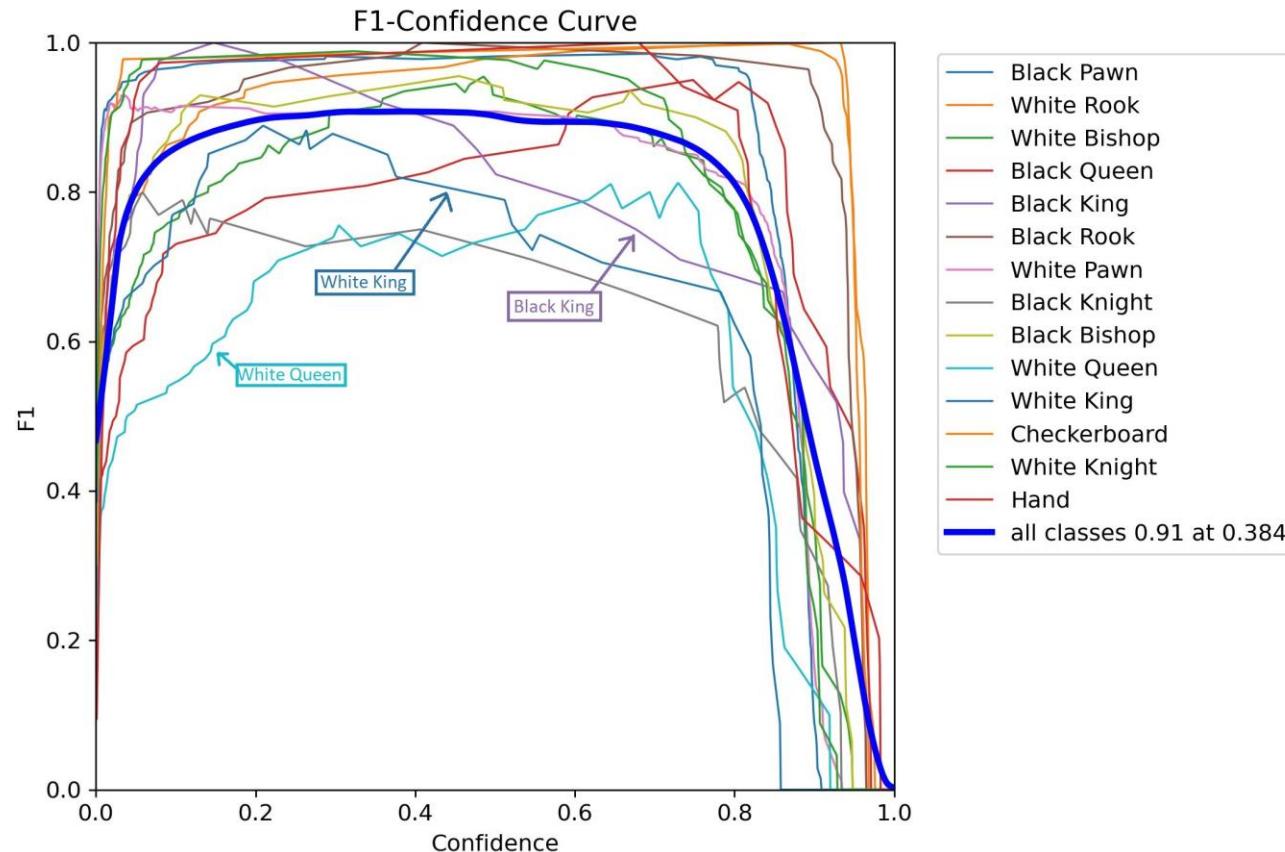
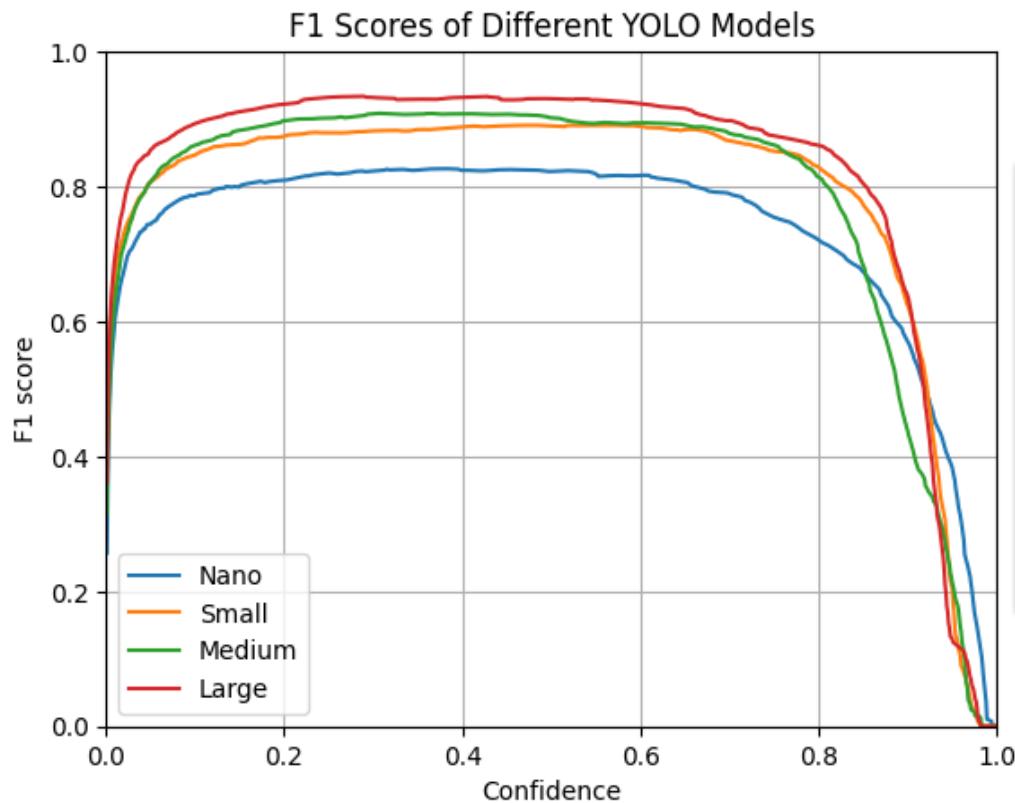


Figure: F1 Score of medium model

# YOLO Model Selection

## Test: F1 Scores

- Confidence threshold is determined by F1 Score.
- Medium model is selected.



Model	Max F1 S.	Confidence
Nano	0.81	0.374
Small	0.89	0.553
Medium	0.91	0.384
Large	0.93	0.274

# YOLO Model Selection

- Some detection examples: Only pawns



# YOLO Model Selection

- Some detection examples: Rooks and Bishops



# YOLO Model Selection

- Some detection examples: Knights, Kings and Queens



# YOLO Model Selection

- Some detection examples: Checkerboard and Hands



# YOLO Model Selection

Detection Challenges:

- White bishops are hard to detect.
- Queens and kings are sometimes confused for each other.

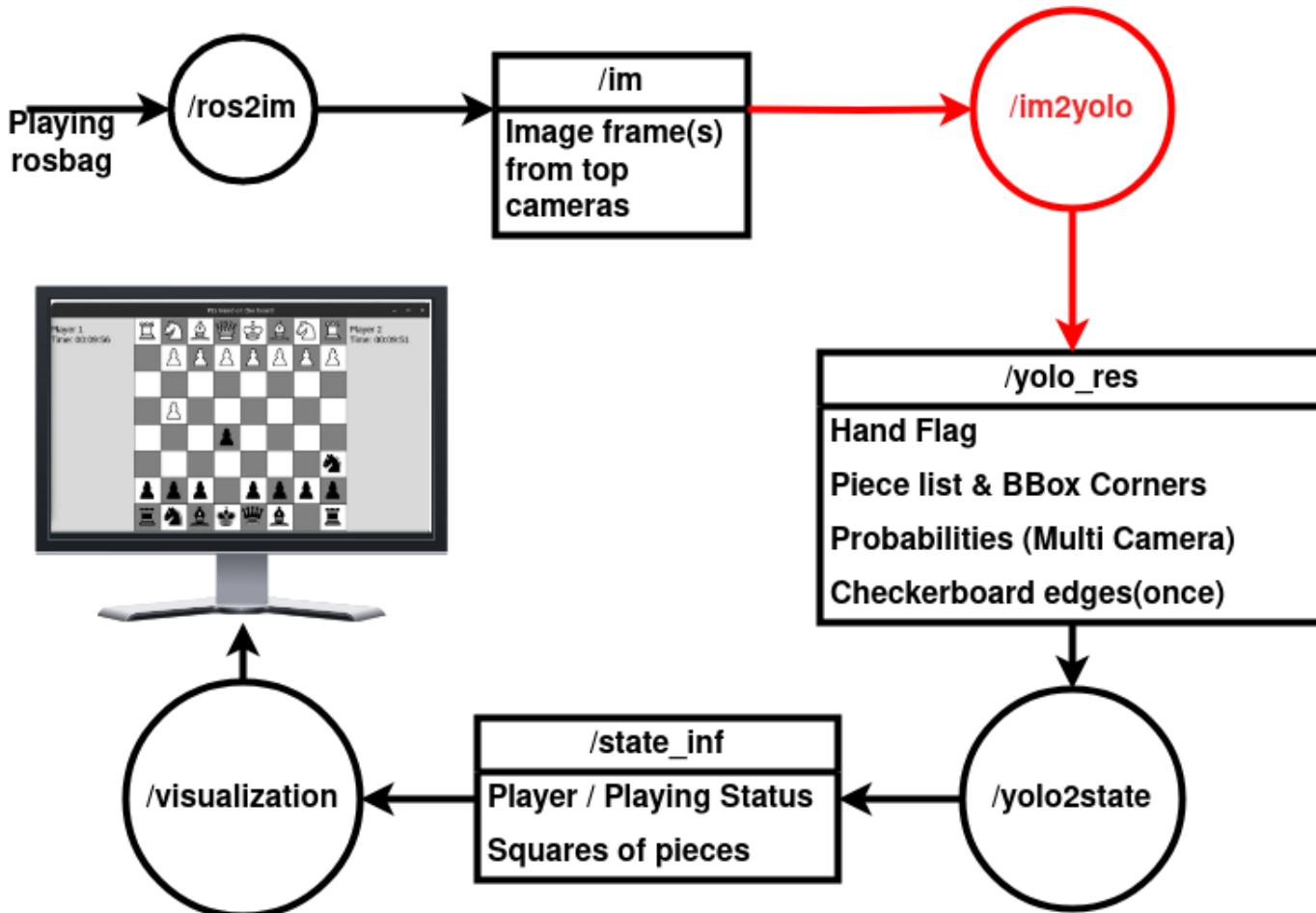


Figure 1: Wrongly detected white queen

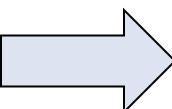


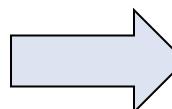
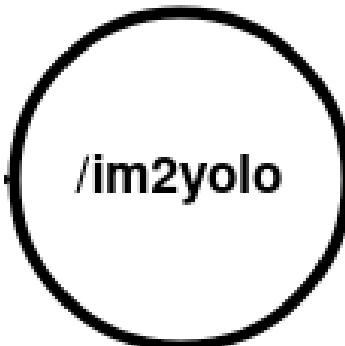
Figure 2: Undetected white bishop

# Piece, Checkerboard and Hand Detection



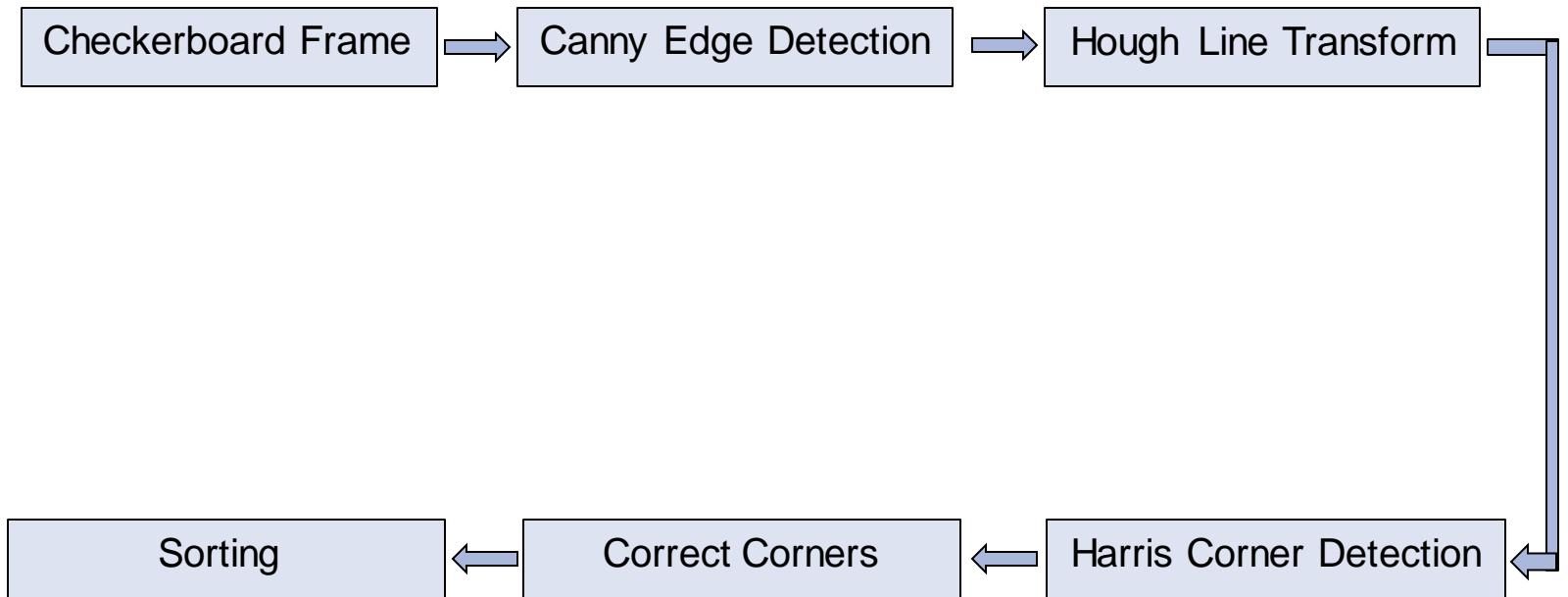
# Piece, Checkerboard and Hand Detection

Frame(s) 



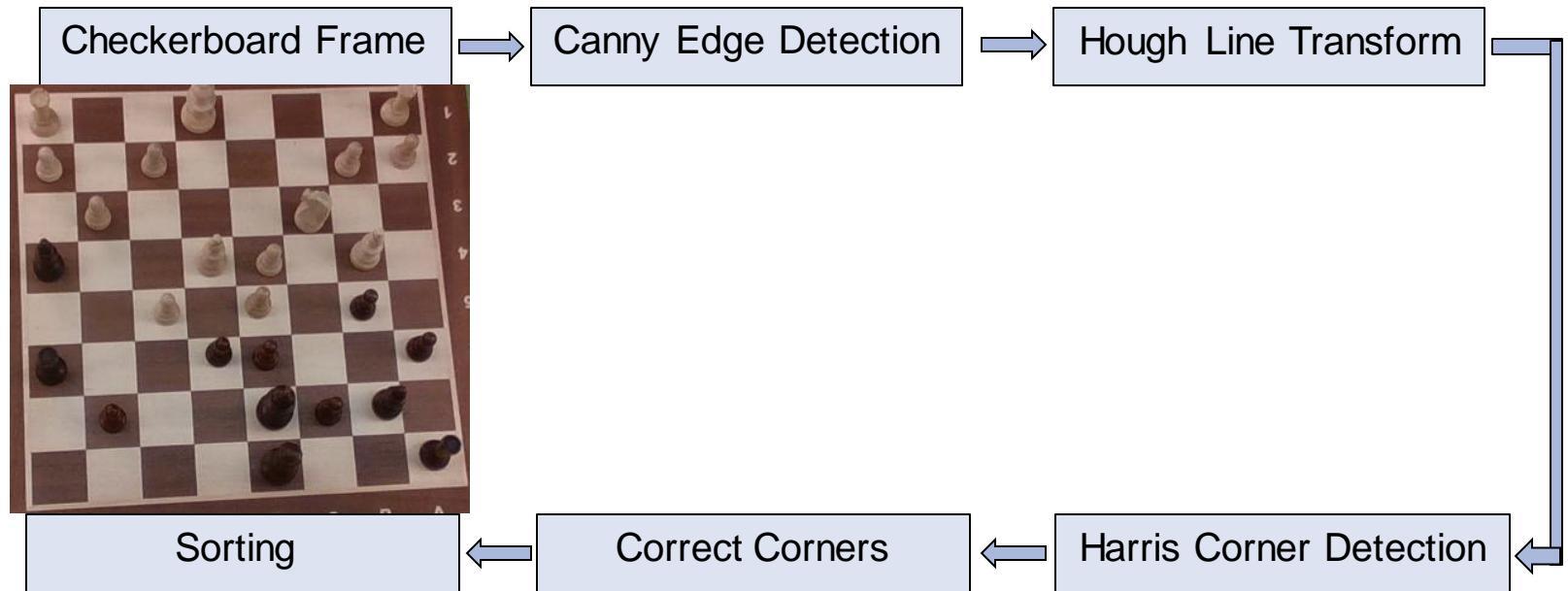
- Checkerboard coordinates
- Hand flag
- Piece classes
- Piece bounding boxes
- Piece probabilities

# Checkerboard Corner Detection



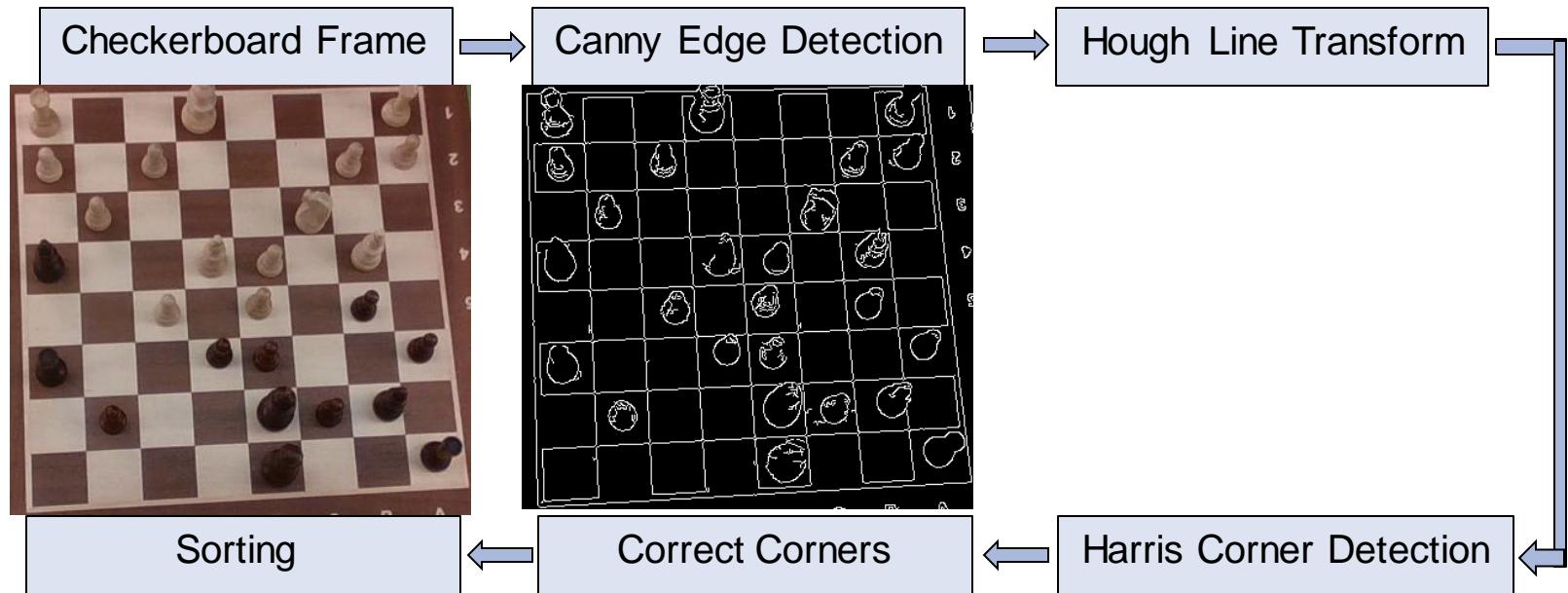
Sorted  
corner  
coordinates

# Checkerboard Corner Detection



Sorted  
corner  
coordinates

# Checkerboard Corner Detection



Sorted  
corner  
coordinates

# Checkerboard Corner Detection

- Hough-Line Transform:
  - Point -> Sinusoid
  - Intersection of sinusoids means line

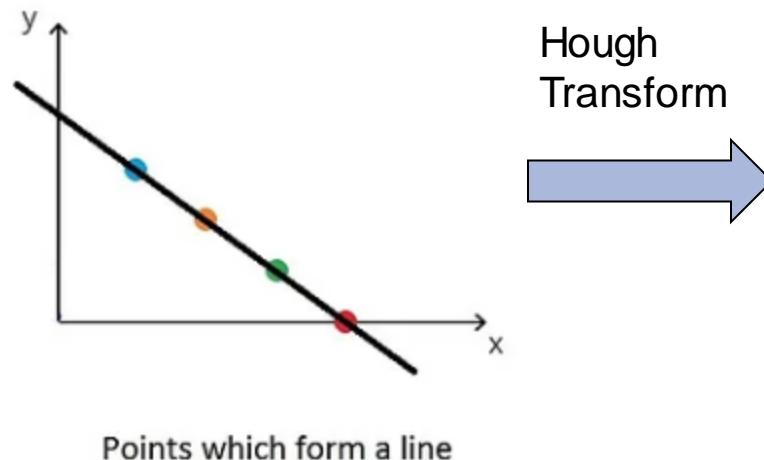


Figure 1: Line on image domain. Adapted from <https://shorturl.at/lBOX5>

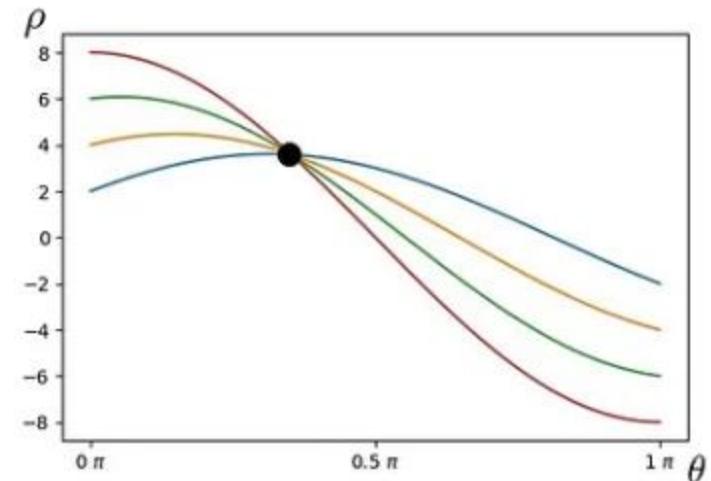
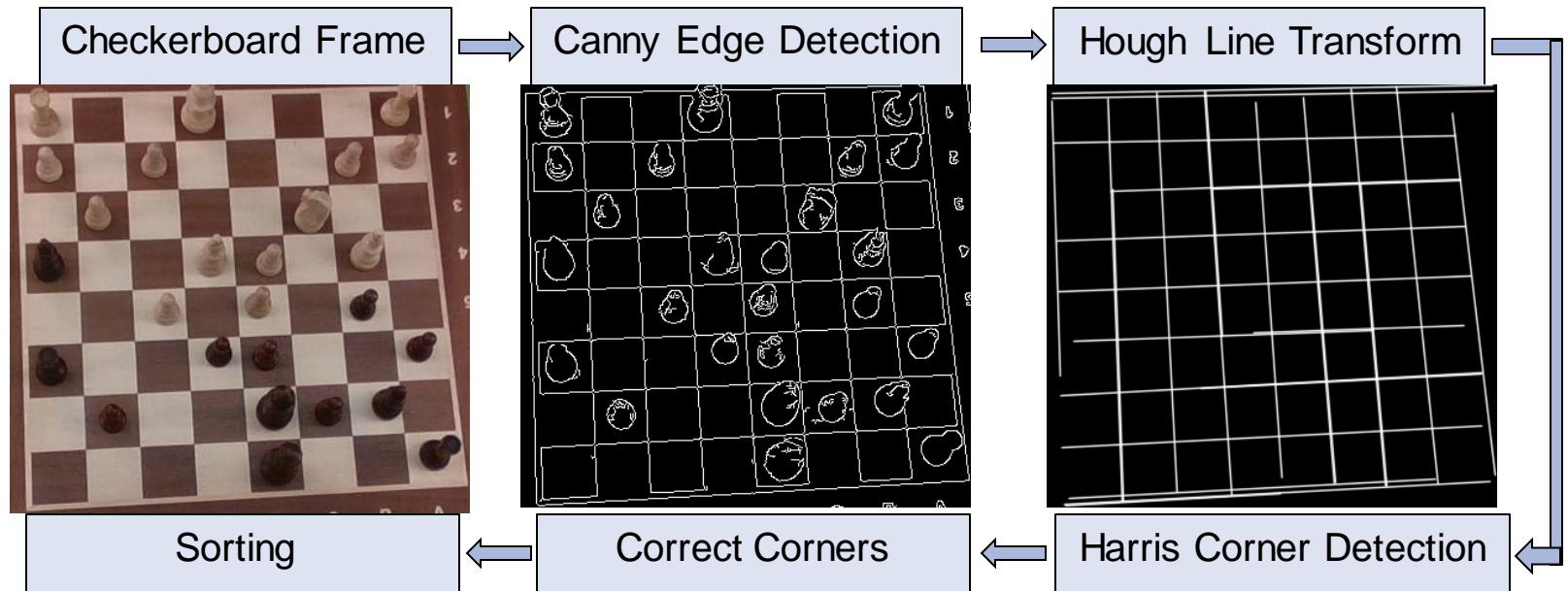


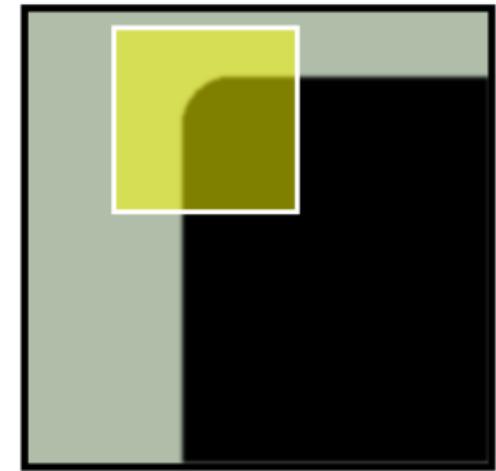
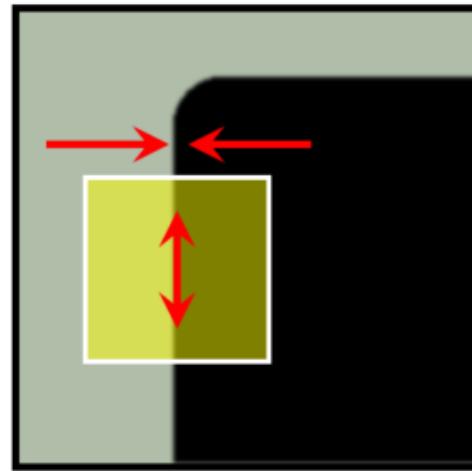
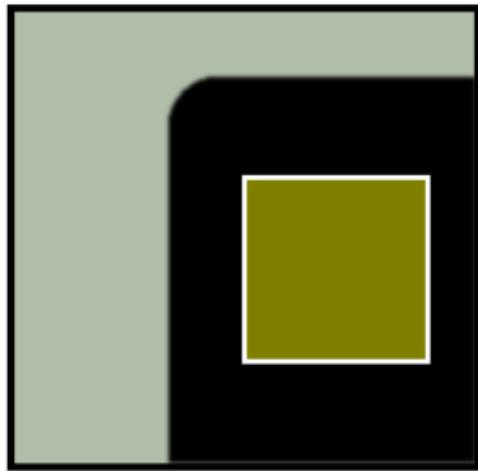
Figure 2: Sinusoids on image domain. Adapted from <https://shorturl.at/lBOX5>

# Checkerboard Corner Detection



Sorted  
corner  
coordinates

# Harris Corner Detection: Basic Idea



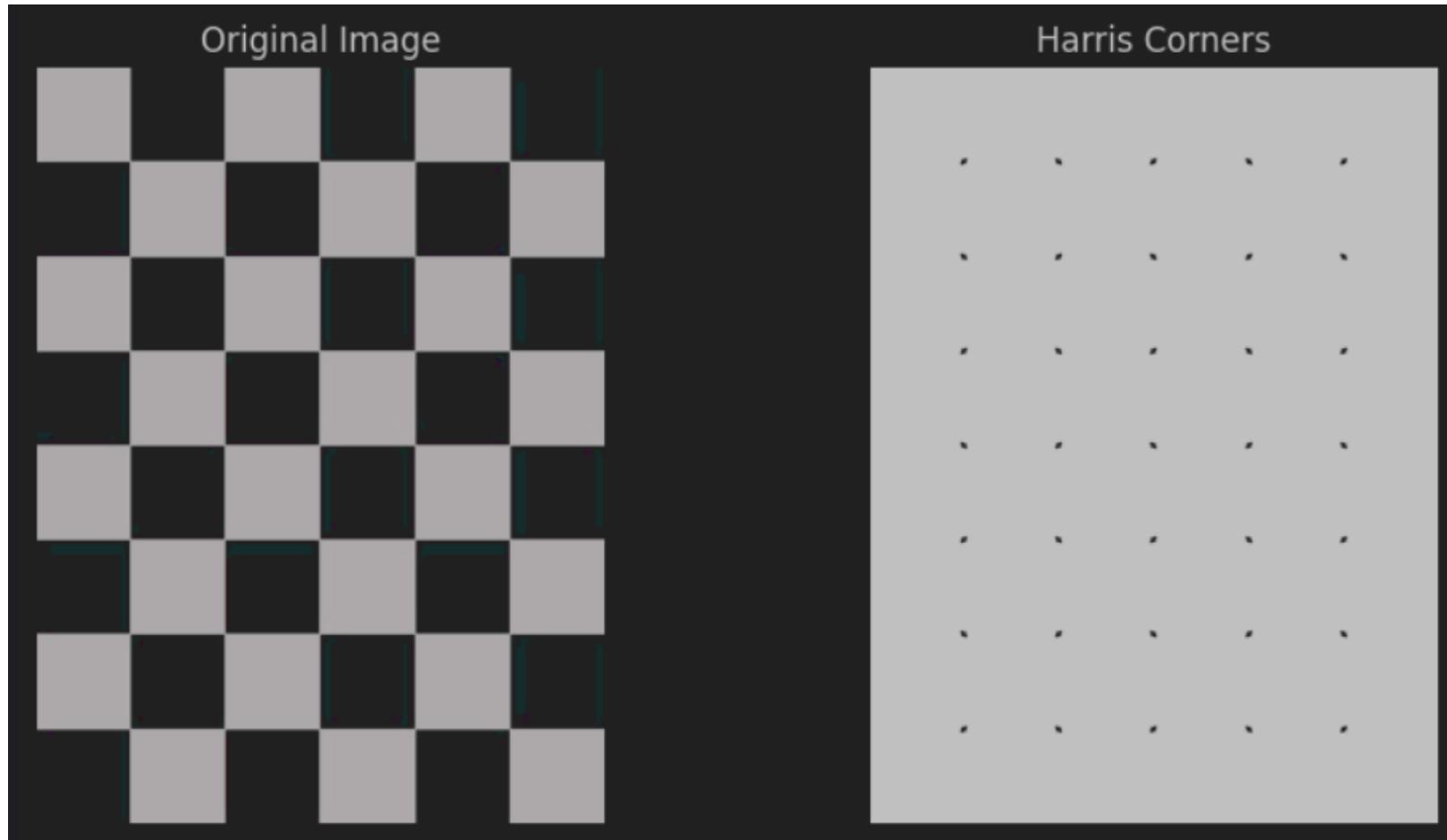
**"flat"** region:  
no change in  
all directions

**"edge"** : no change  
along the edge  
direction

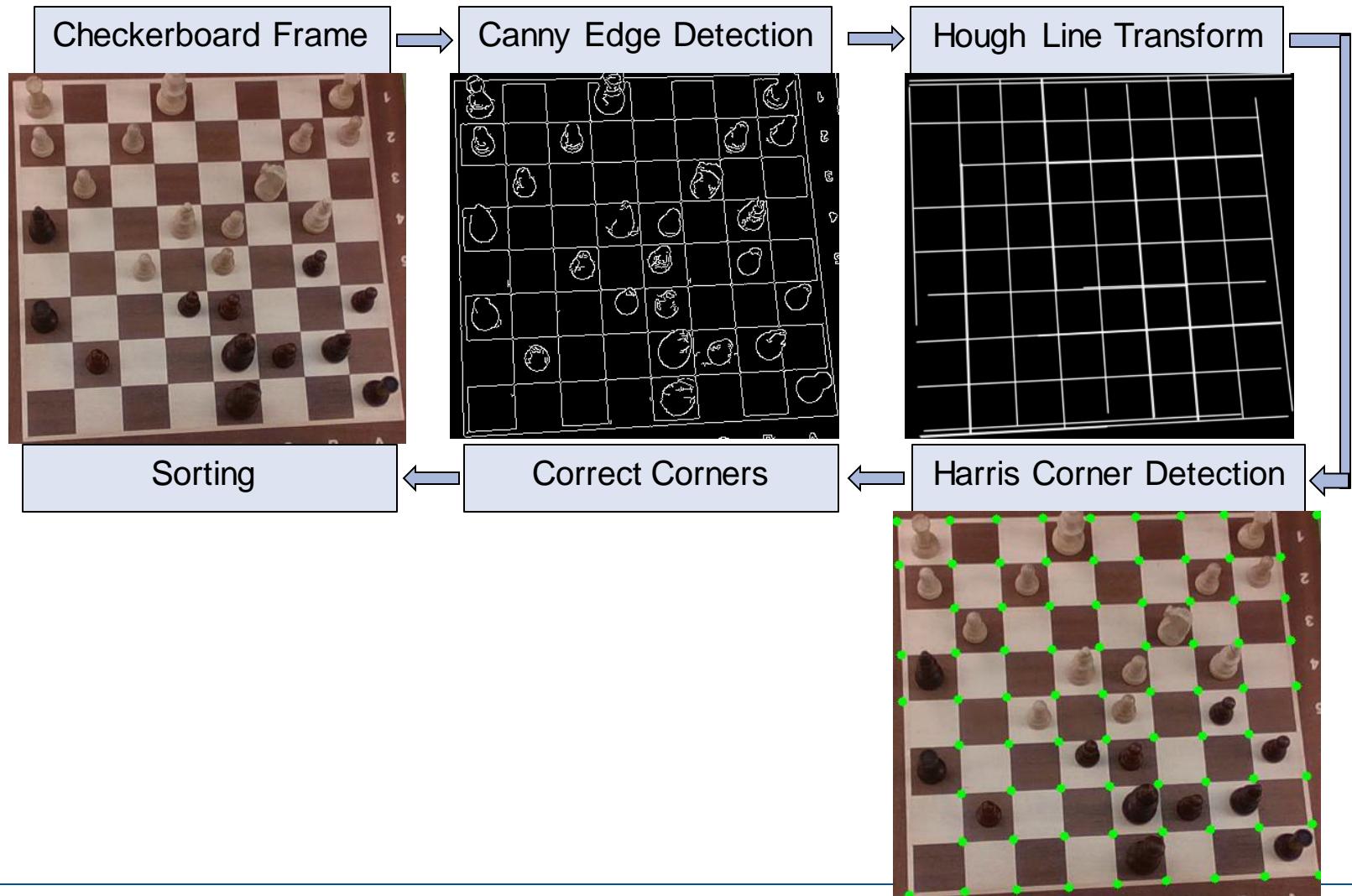
**"corner"** : significant  
change in all directions  
with small shift

Figure: Harris Corner Detection. Adapted from  
<https://shorturl.at/jtwQ8>

# Harris Corner Detection: Good for chess



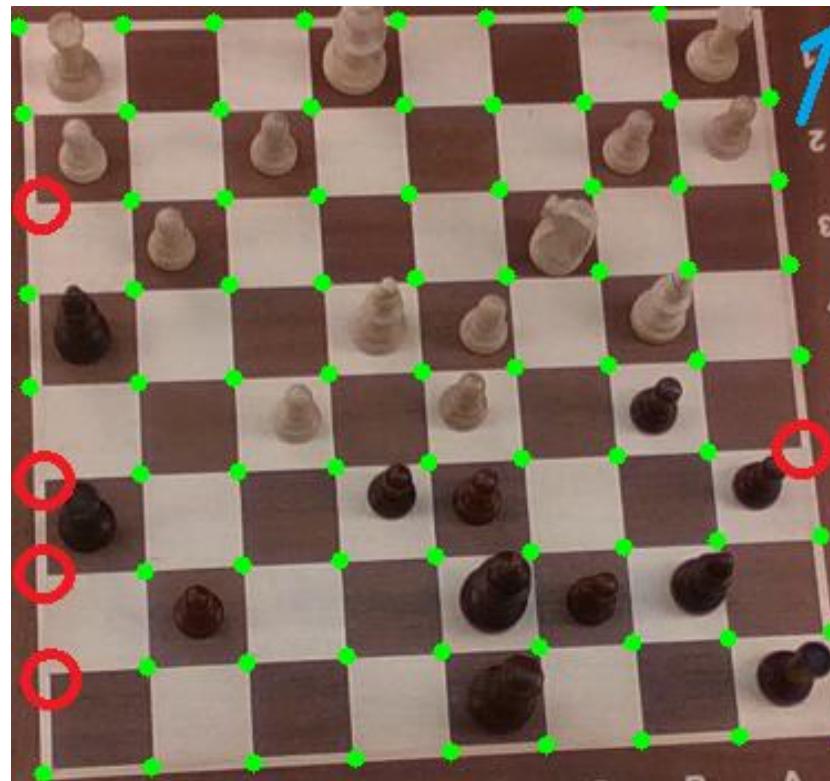
# Checkerboard Corner Detection



Sorted  
corner  
coordinates

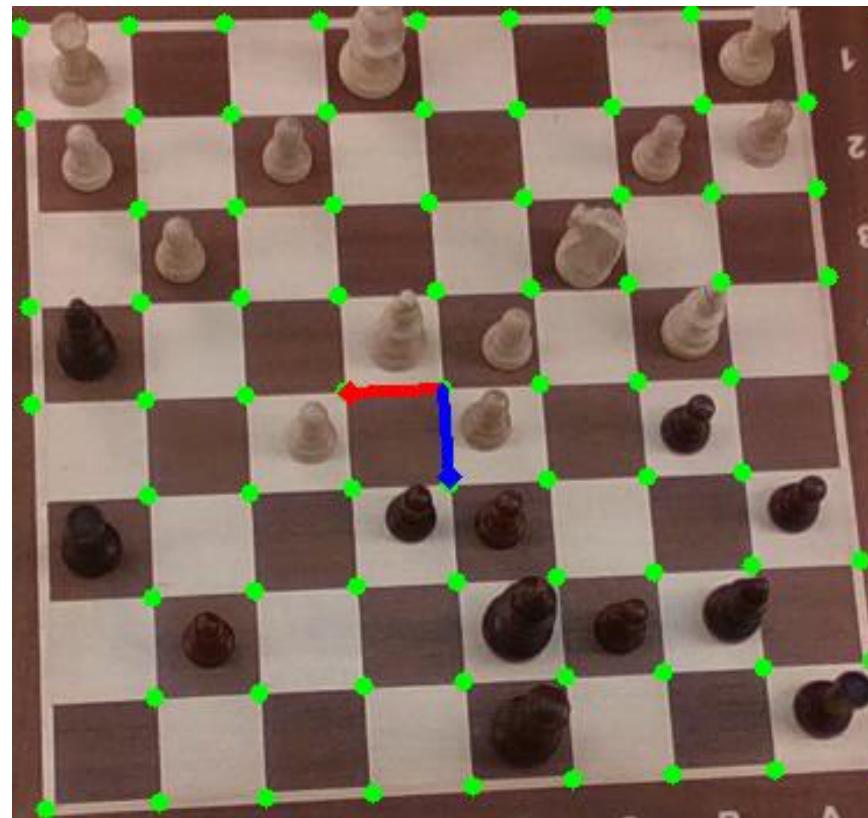
# Checkerboard Corner Detection

- What if some corners are not detected? Or some corners are wrongly detected?



# Checkerboard Corner Detection

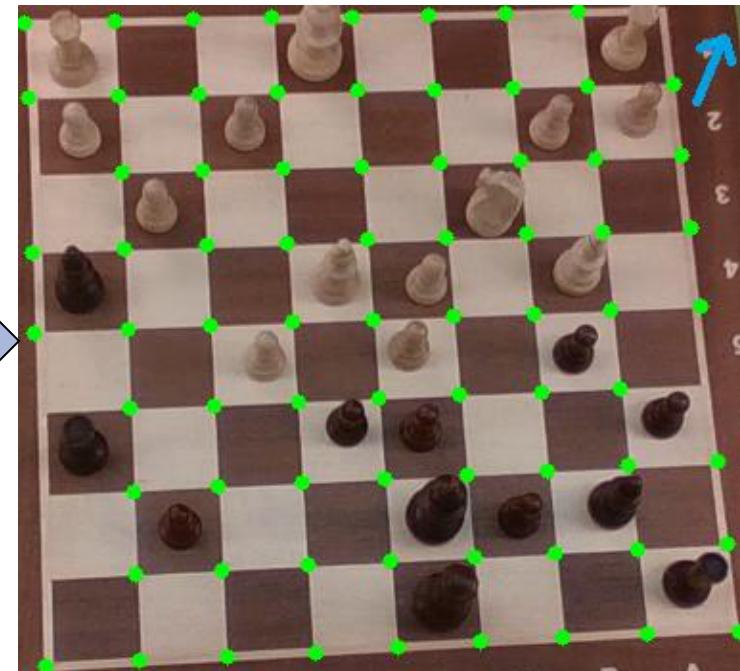
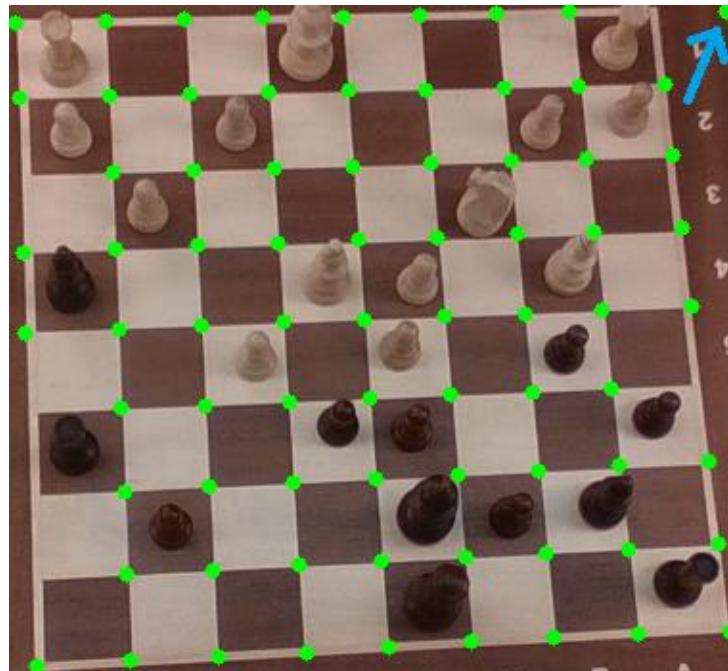
- Predict vertical and horizontal edge vectors.
- Median of existing vertical and horizontal vectors.



# Checkerboard Corner Detection

For excess corners

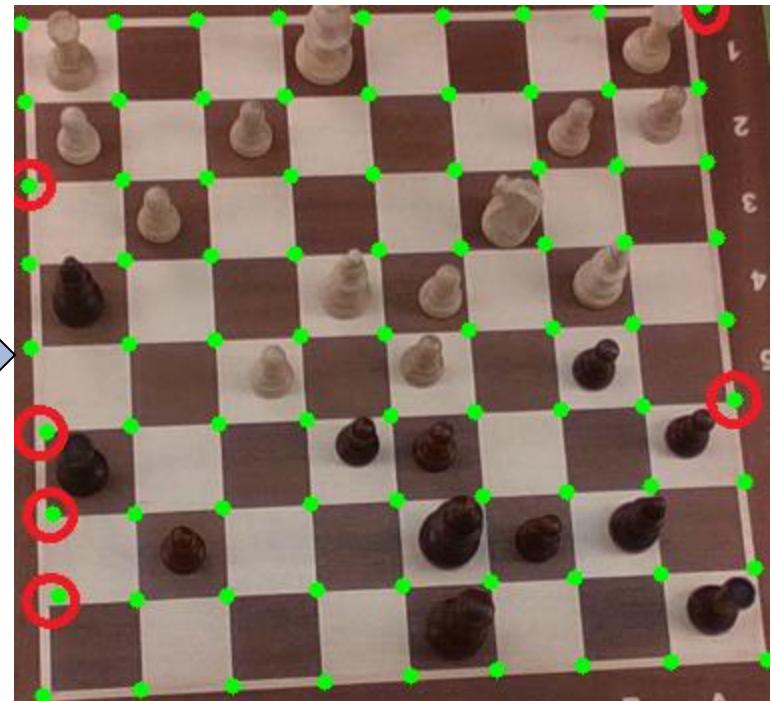
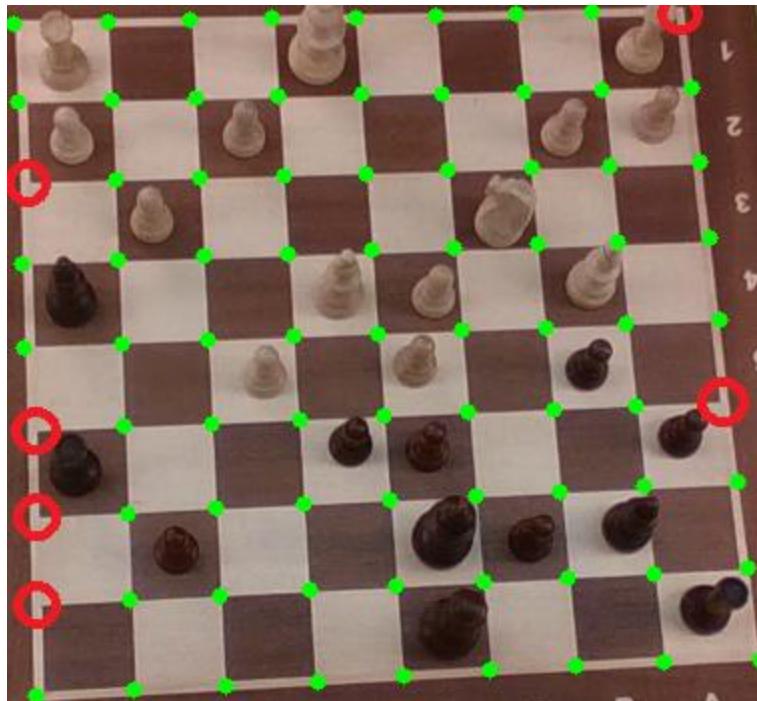
- Look at detected corner locations
- If detected corner does not match with any of predicted corners, delete the corner.



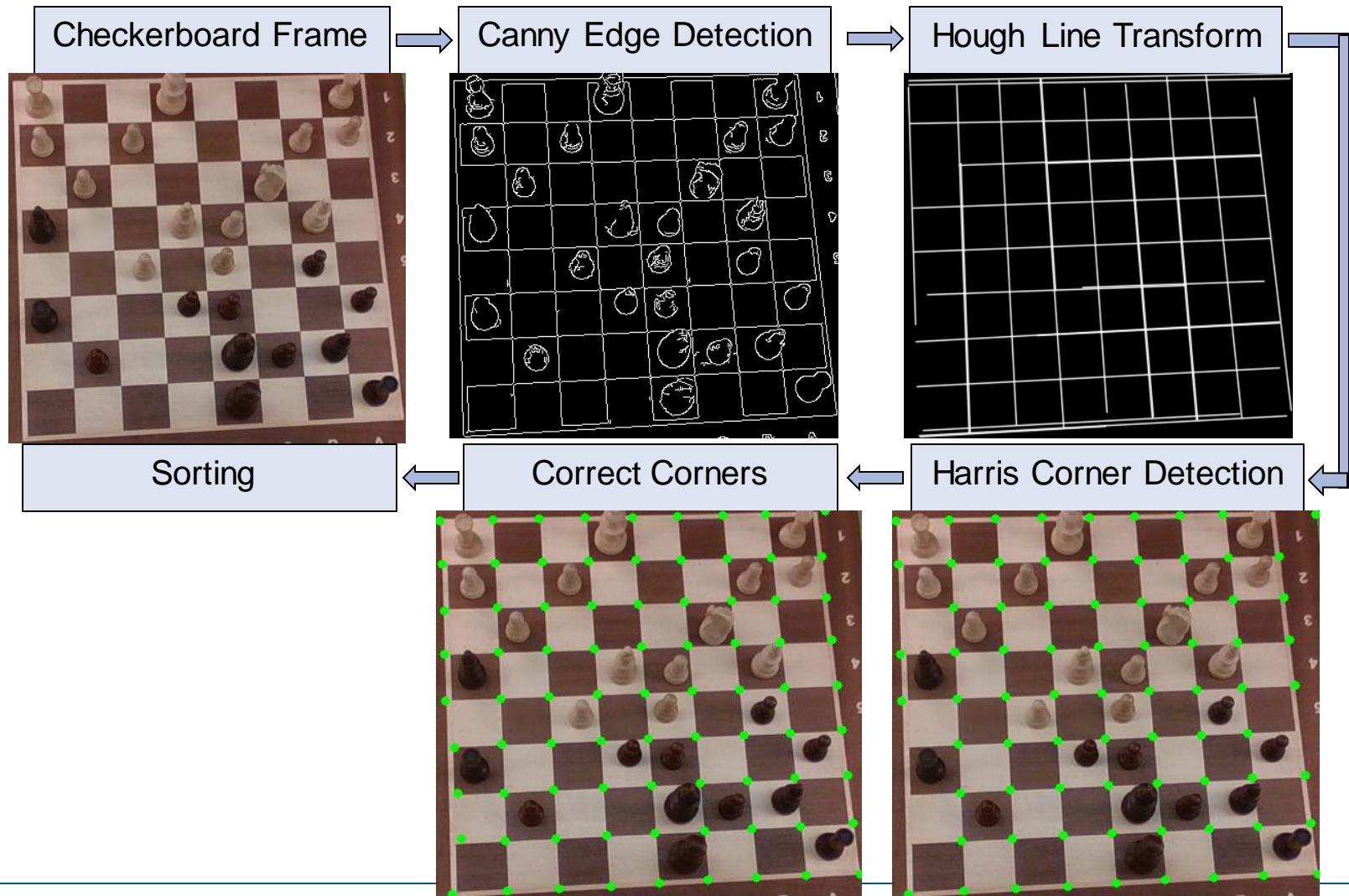
# Checkerboard Corner Detection

For missing corners

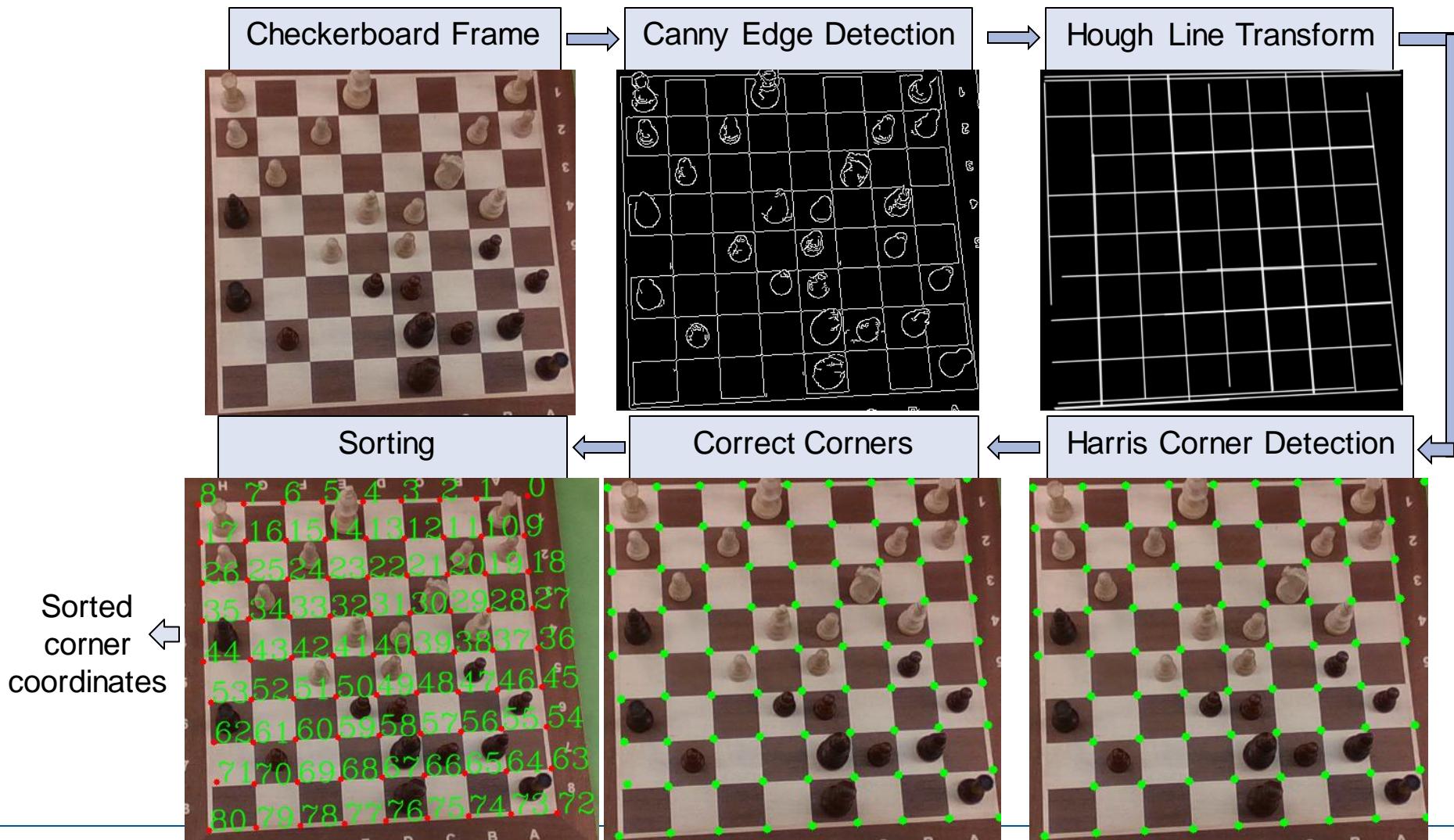
- Look at expected corner locations.
- If corner exists pass, if corner does not exist add a corner.



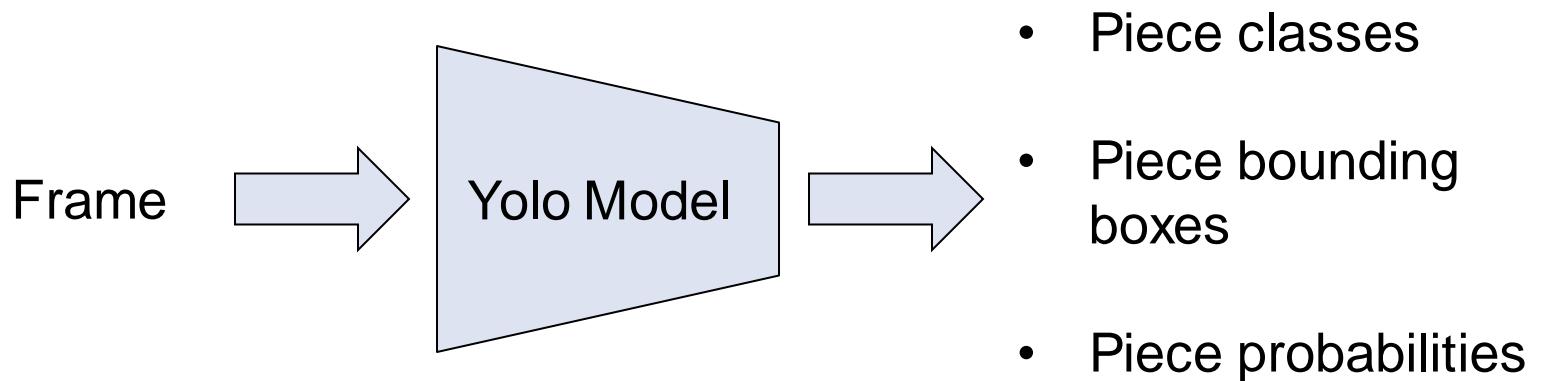
# Checkerboard Corner Detection



# Checkerboard Corner Detection

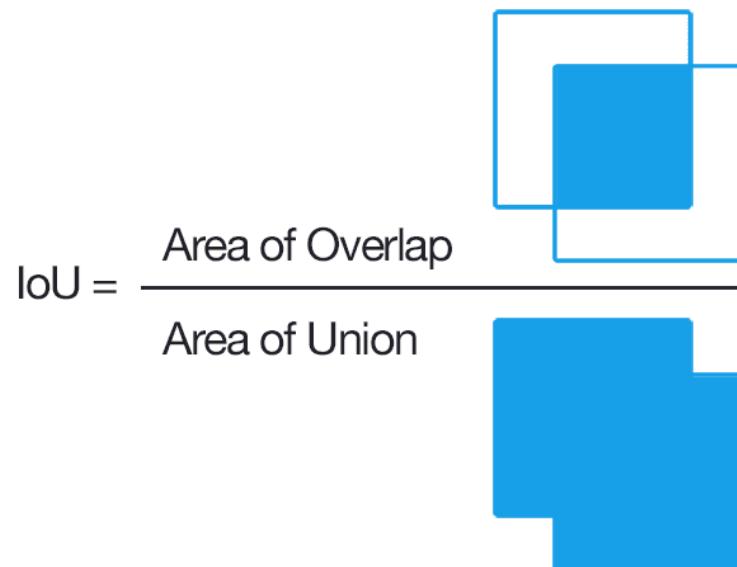


# Piece Detection



# Hand Detection: Intersection Over Union

- Hand Flag = 
$$\begin{cases} 1 & \text{if } \text{IoU} > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$



# Intersection over Hand (IoH)

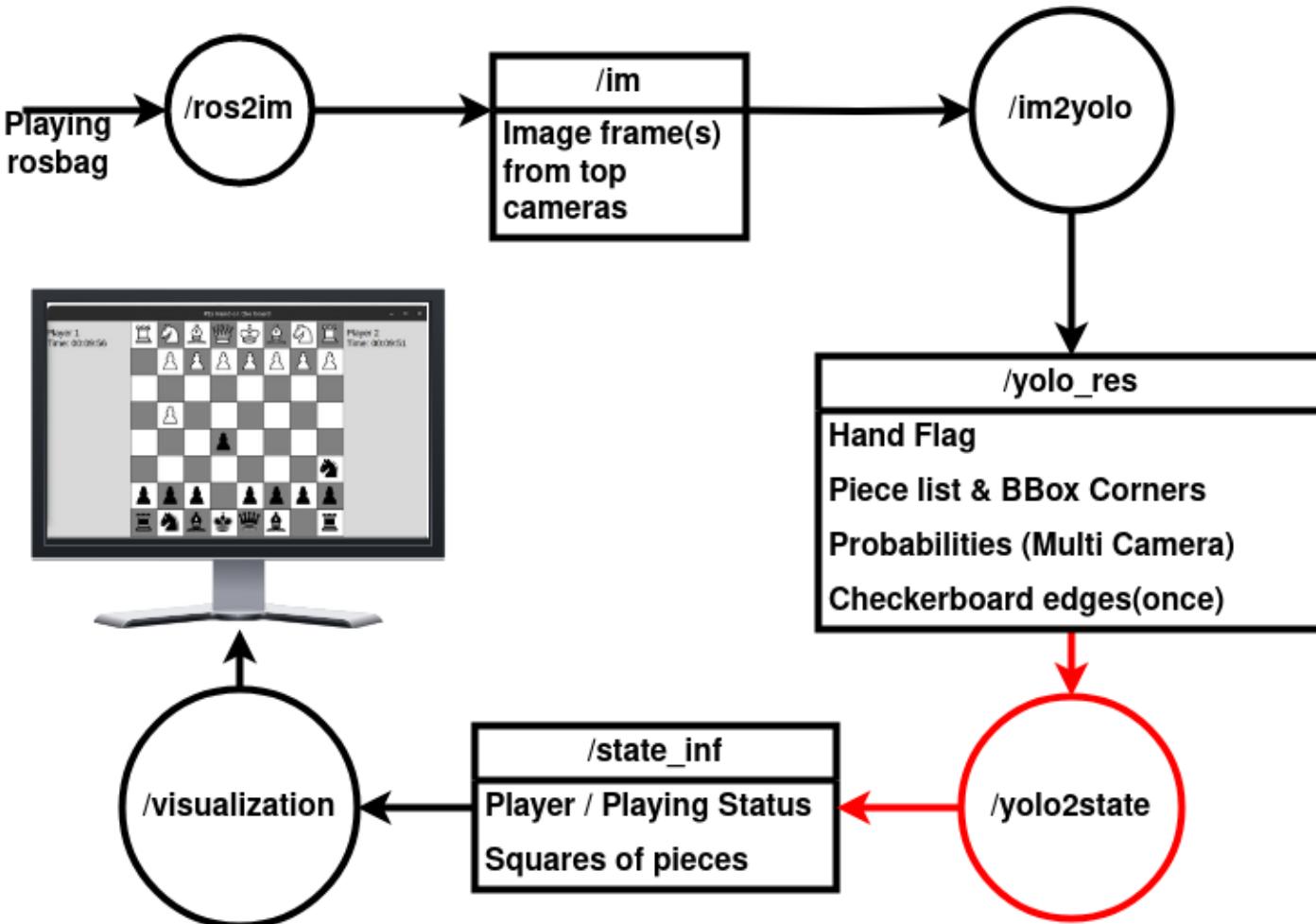
To understand what ratio of hand is over the checkerboard





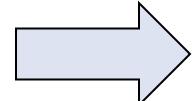
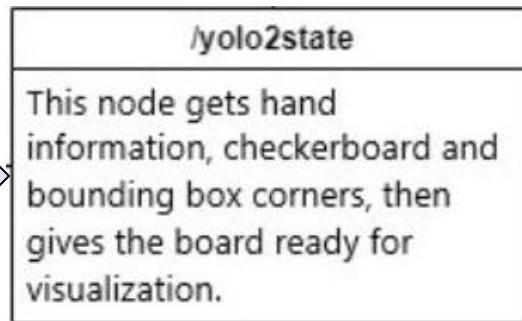
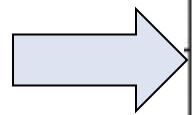
# GAME STATE

# Game State Management



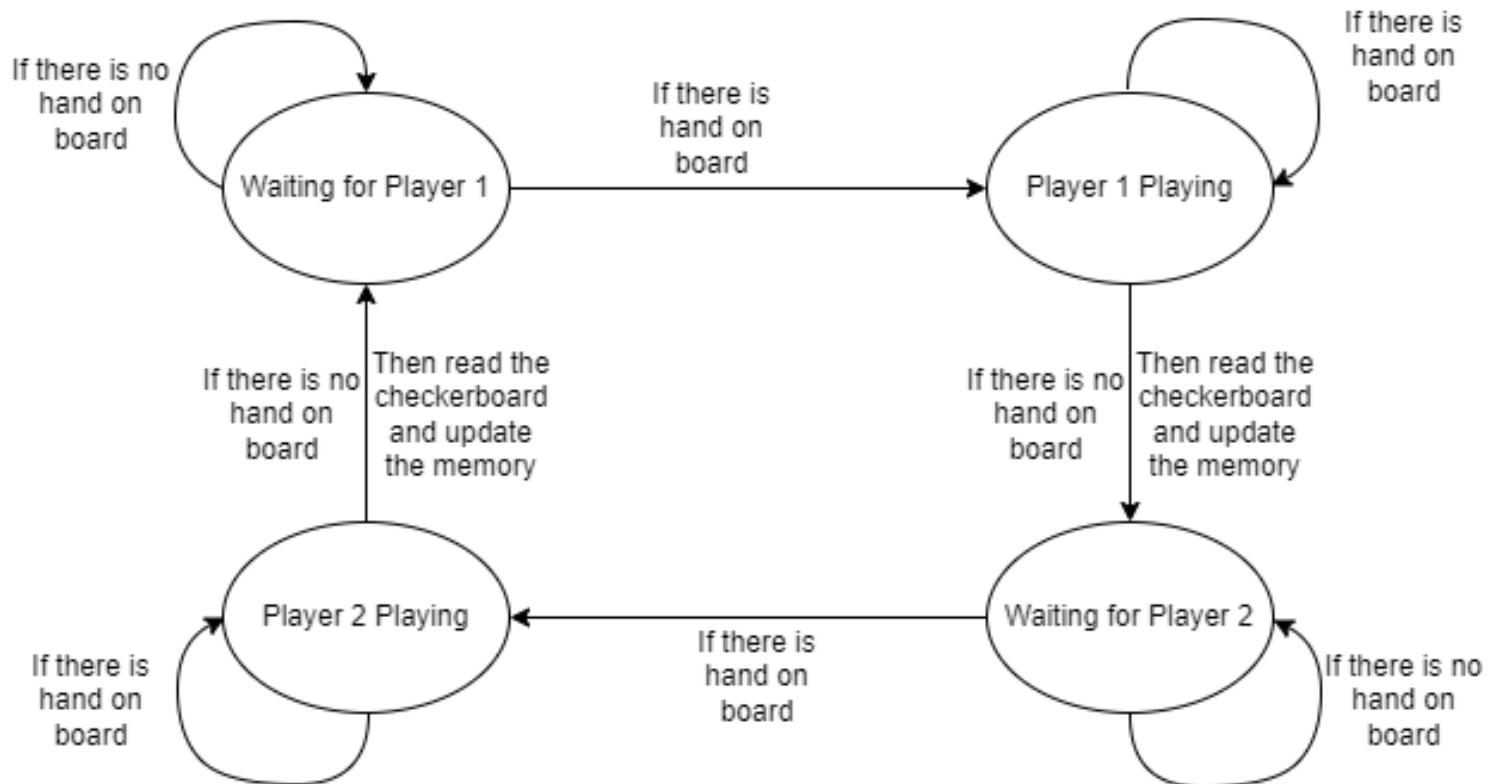
# Game State Management

- Checkerboard coordinates
- Hand flag
- Piece classes
- Piece bounding boxes
- Piece probabilities



Chessboard  
Player/Playing?

# Game State Management – State Machine





# Game State Management

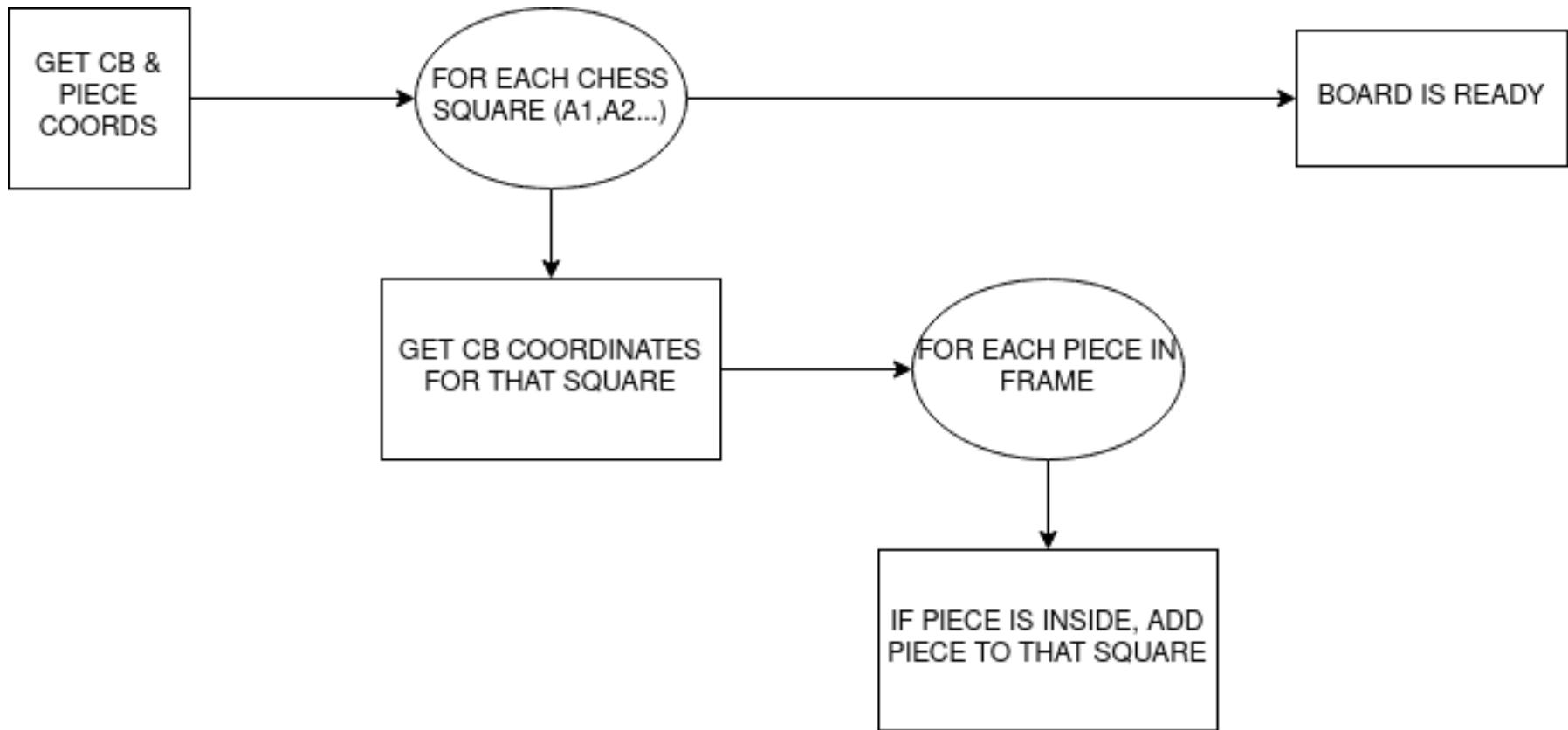
- Read the board only at the end of a move!
  
- The reasons for the decision:
  - Avoid piece detection errors due to blockage of hand
  - Fasten the code



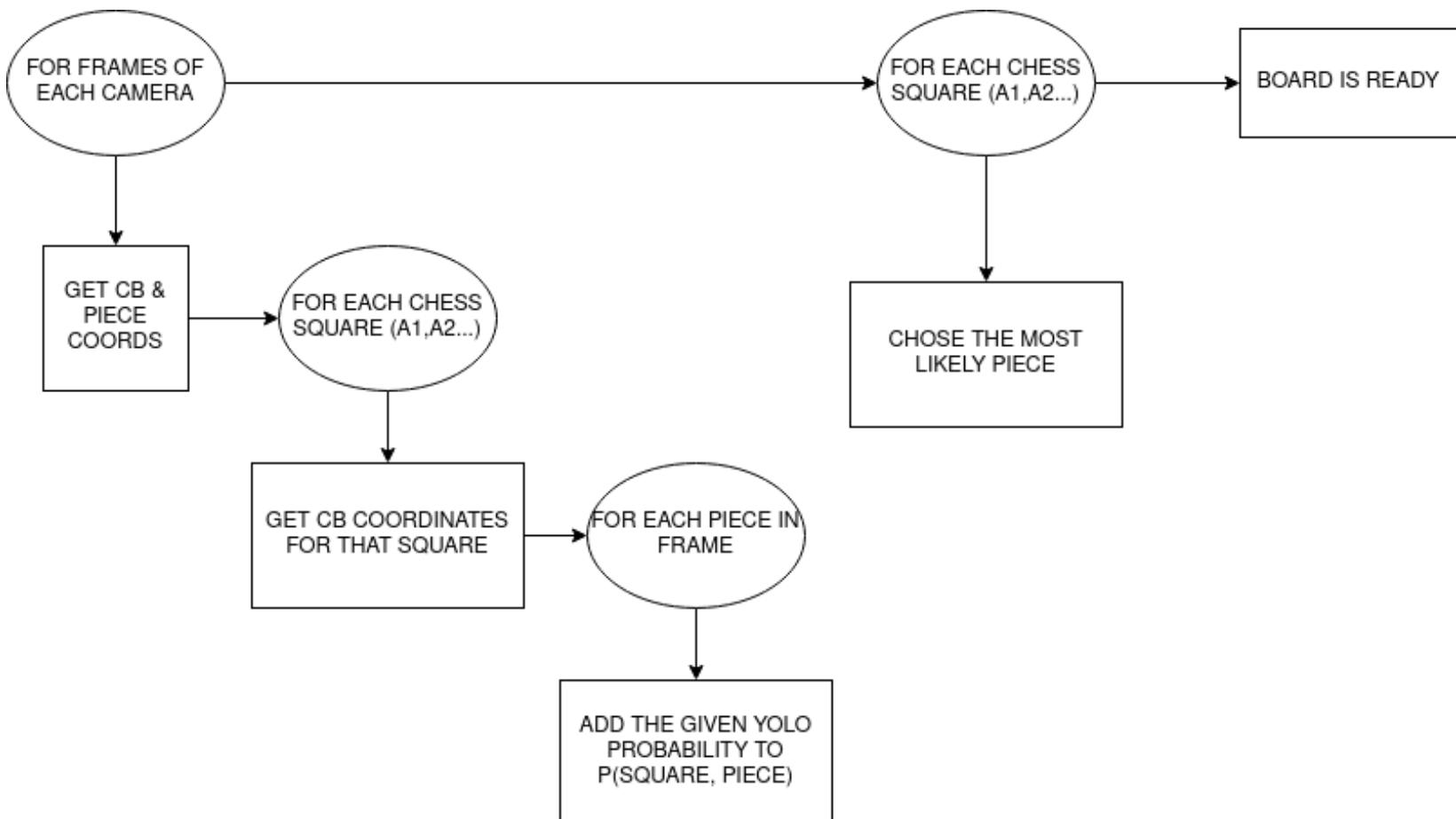
# Placing the Pieces into the Chessboard

- We should evaluate the BBox and Checkerboard corners in order to determine the state of the board.
- The implementation differs for single and multicamera scenarios.

# Chessboard Placement – Single Camera



# Chessboard Placement – Multicamera





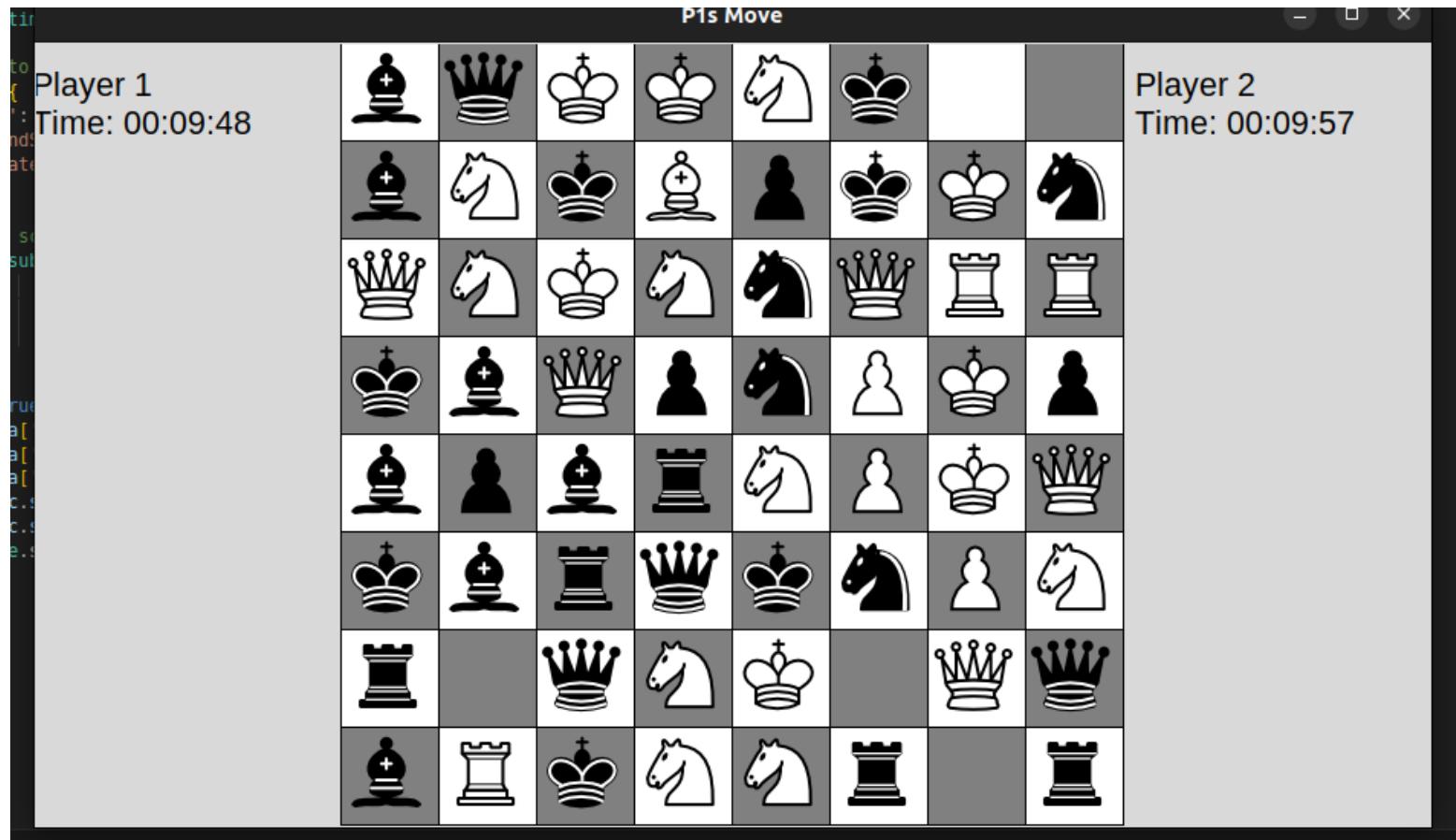
# VISUALIZATION

# Real-time visualization with Tkinter

- Visualization is a subprocess.
- Send game state to subprocess through open pipe.
- Thread checks if data is on pipe and draws it.
- Run in infinite loop

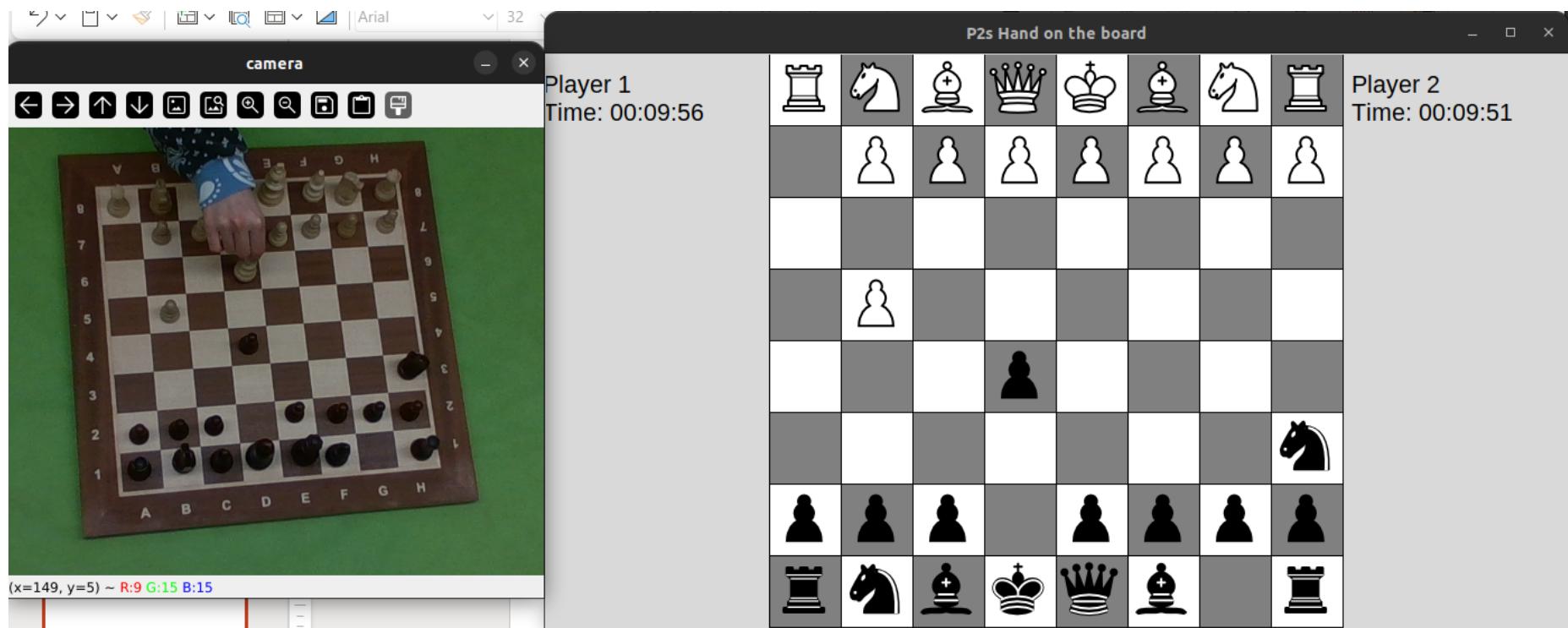


# Real-time tracking of remaining times.



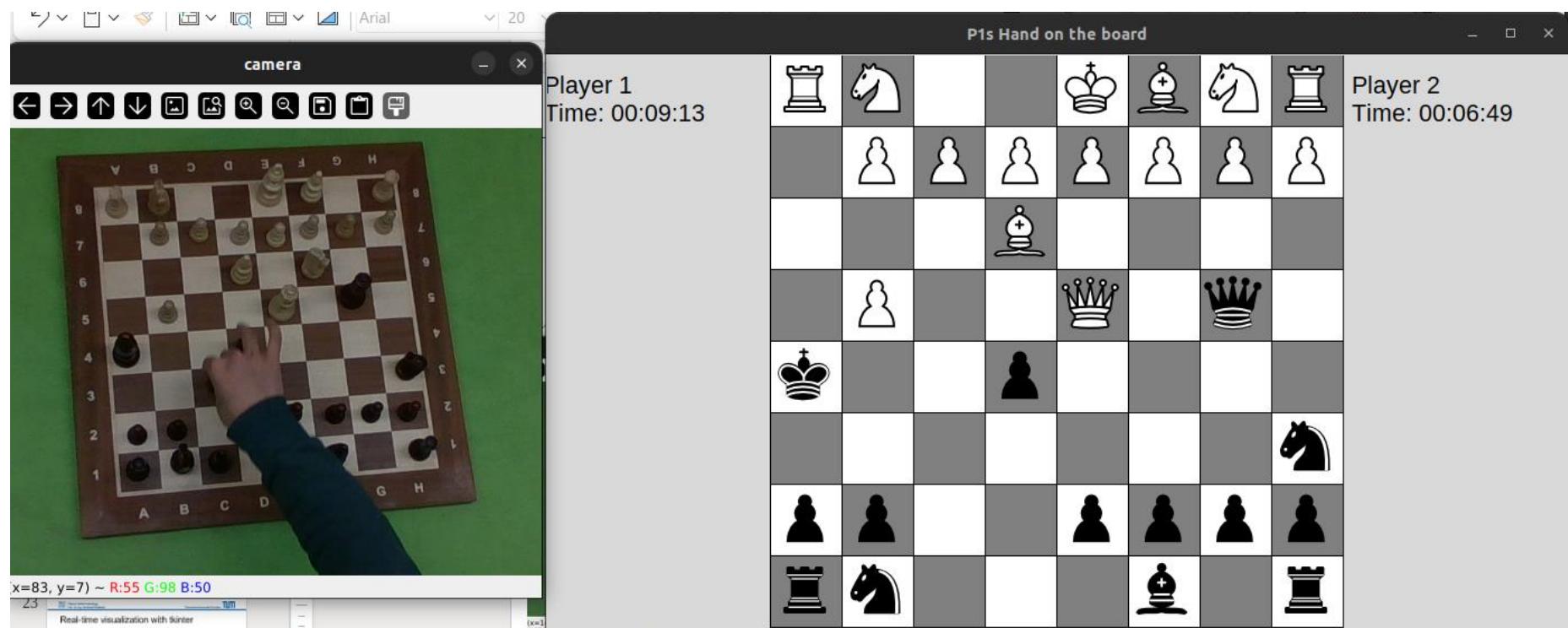
# Hand State on Title

- 4 State: P2s Move, P1s Move, P1s Hand on the board, P2s Hand on the board
- Updated in the Tkinter window in real-time



# Hand State on Title

- 4 State: P2s Move, P1s Move, P1s Hand on the board, P2s Hand on the board
- Updated in the Tkinter window in real-time

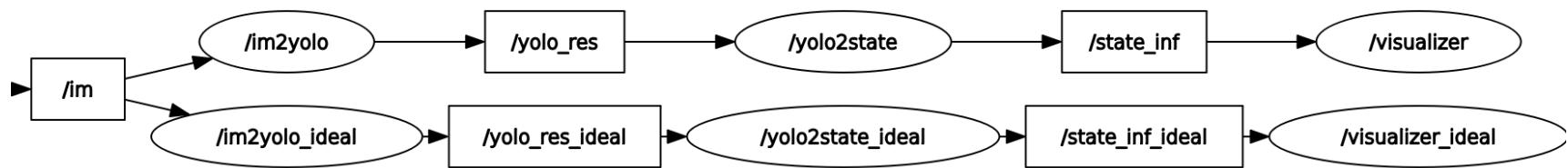




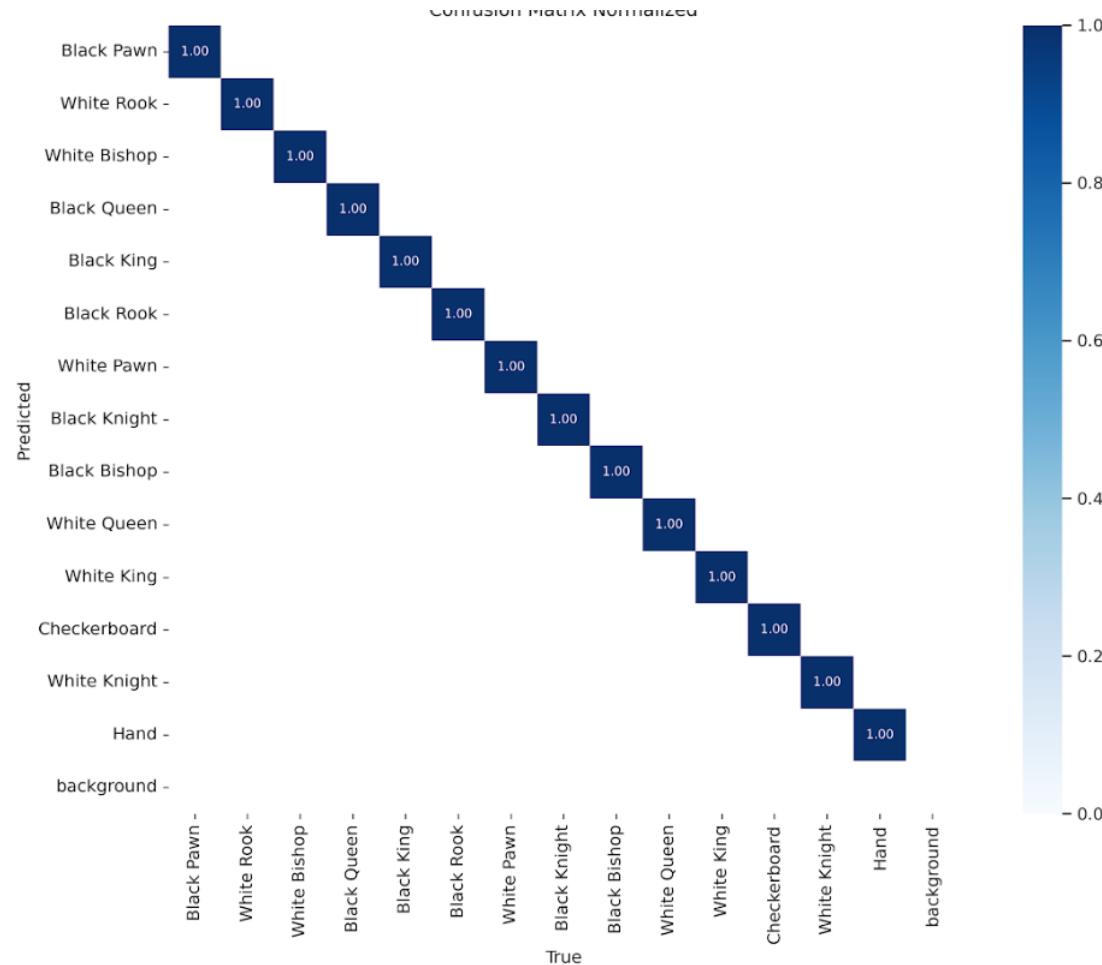
# PERFORMANCE COMPARISONS

# Accuracy calculation on run-time

- Use a perfect YOLO model
  - Every piece is correctly classified
  - Now we can test accuracy by frame

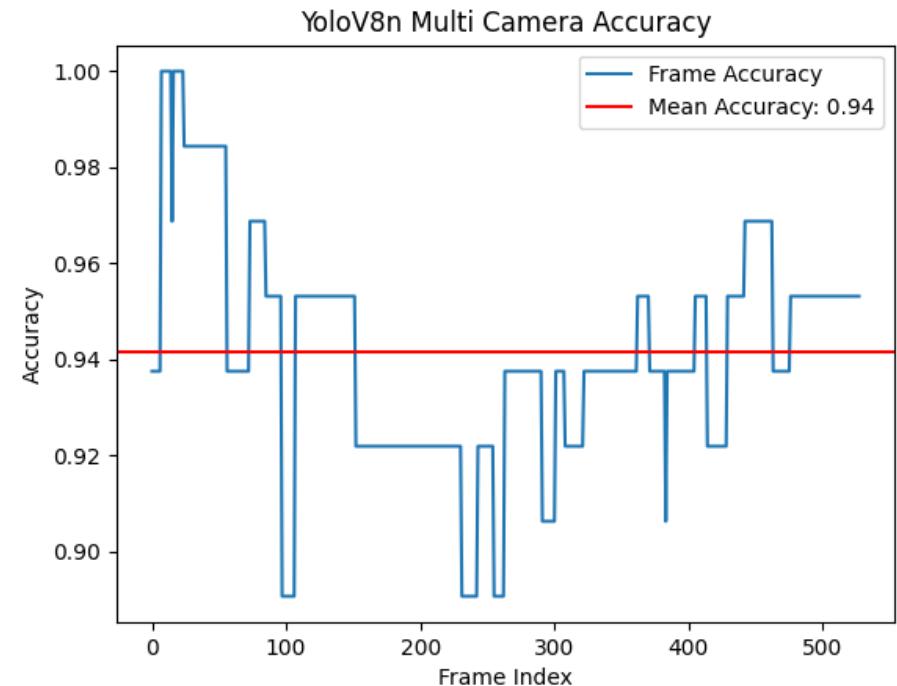
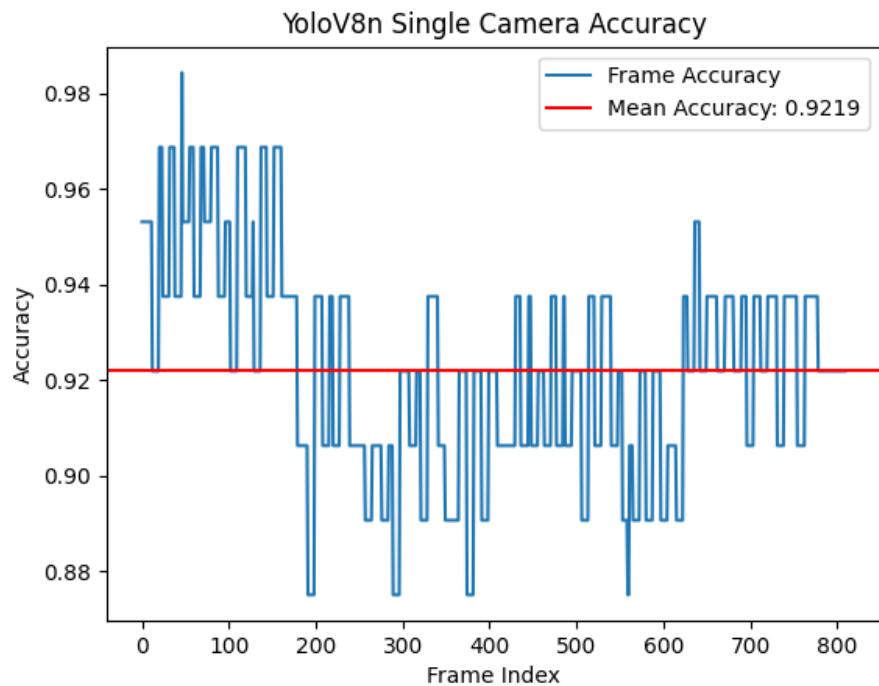


# Confusion Matrix for overfitted model



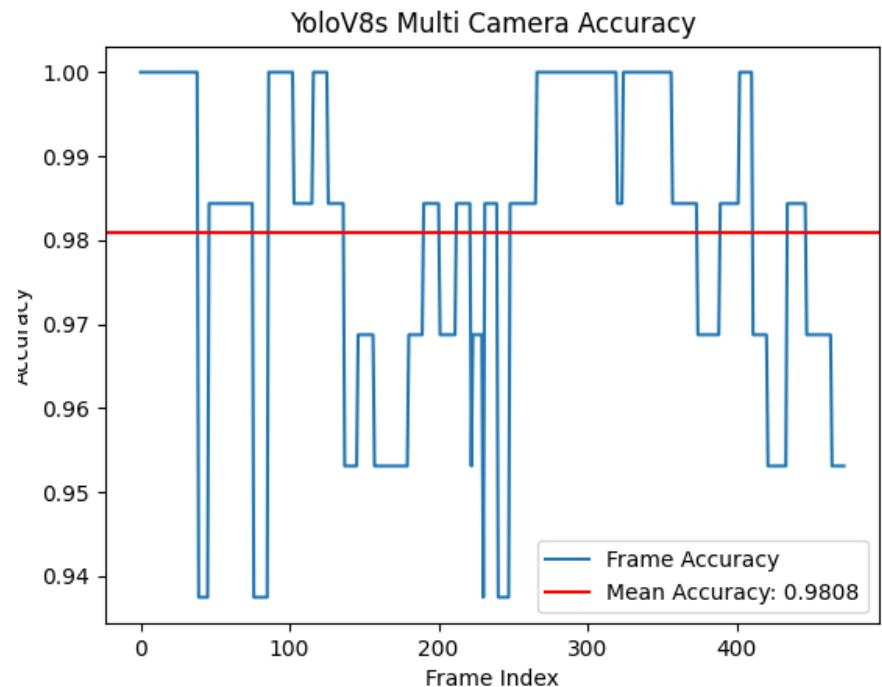
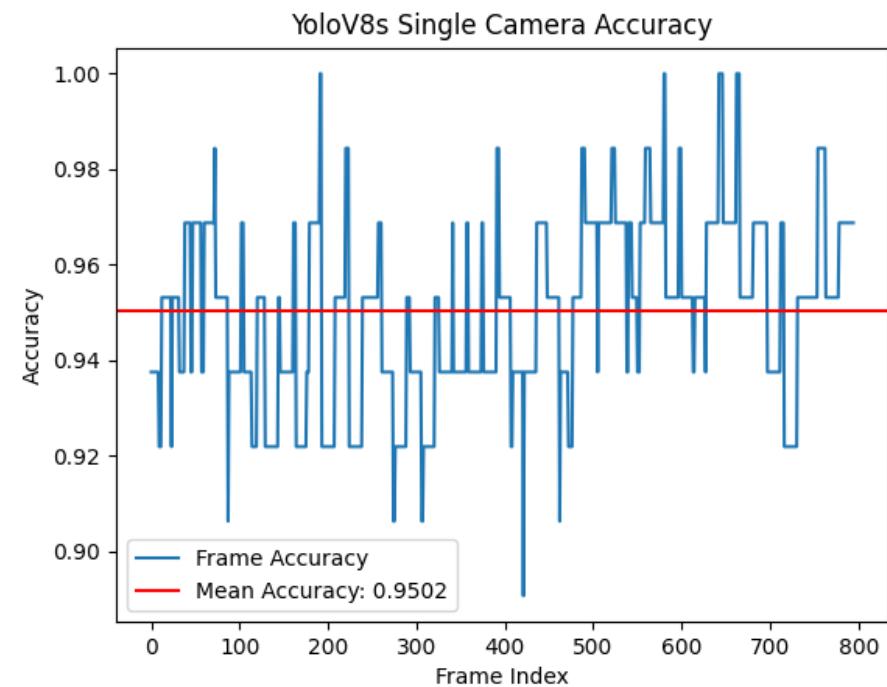
# Accuracy Results for YoloV8n (nano)

- From 0.92 to 0.94 with the cost of delay
- Mean performance is 0.9219



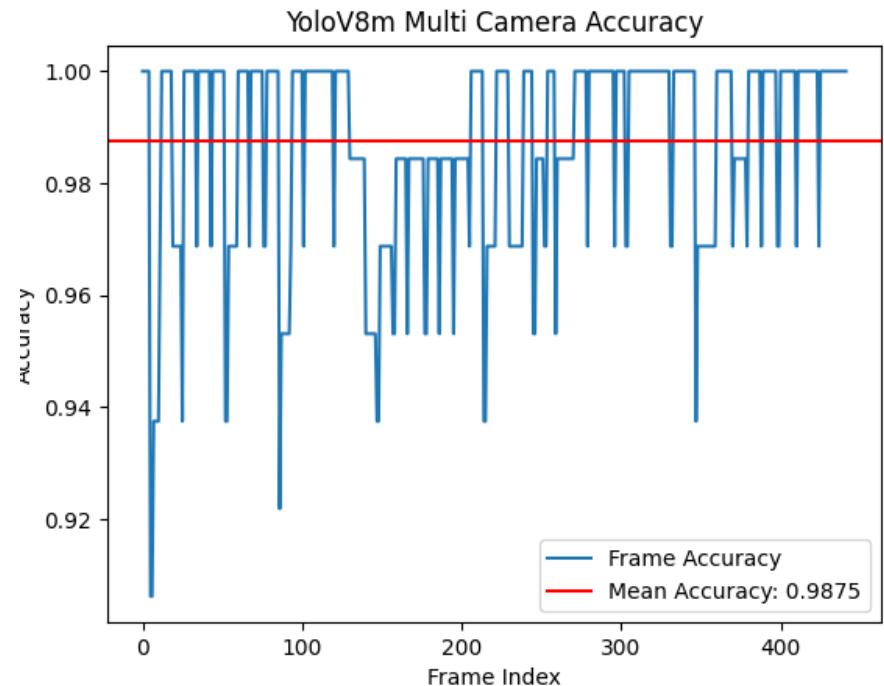
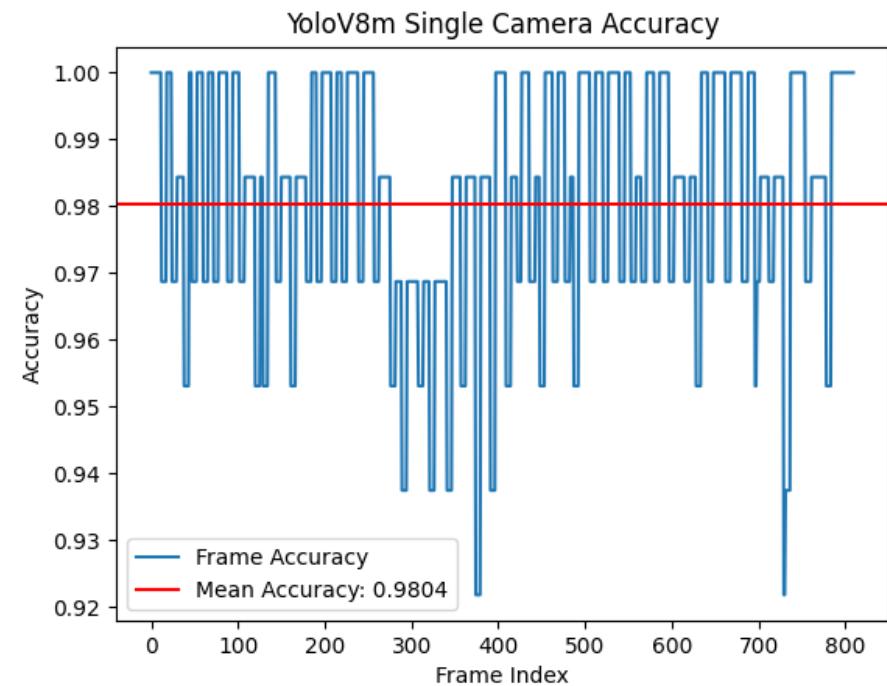
# Accuracy Results for YoloV8s (small)

- Single camera accuracy moved to 0.95 from 0.92 (YoloV8n)
- Multi camera accuracy moved to 0.98 from 0.94 (YoloV8n)



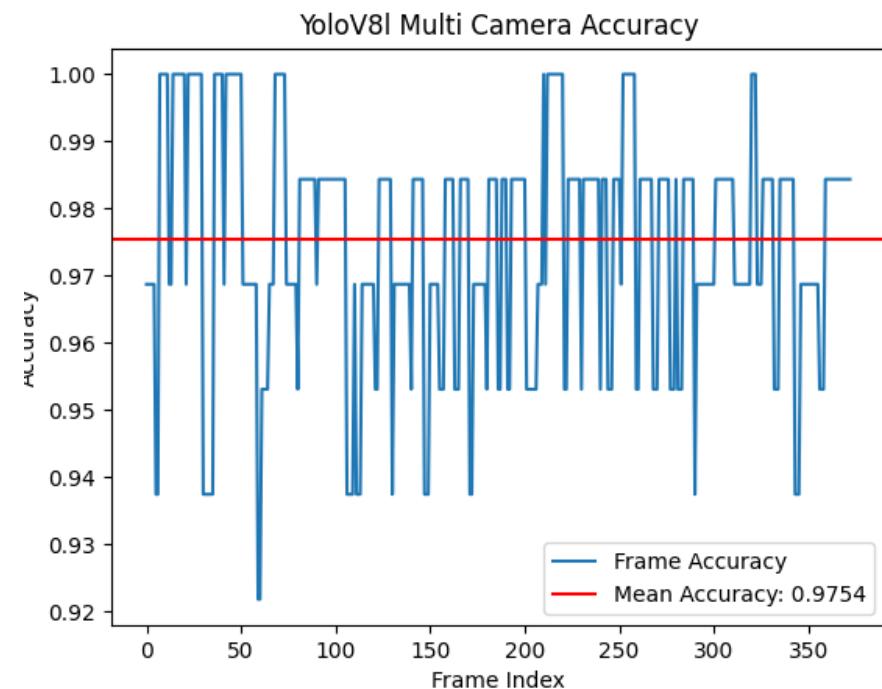
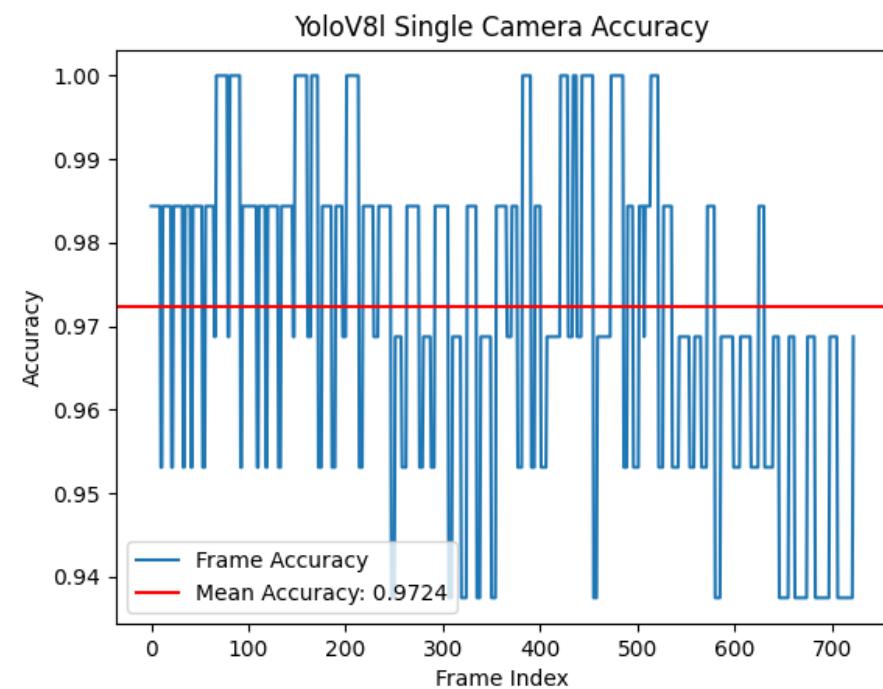
# Accuracy Results for YoloV8m (medium)

- Single camera accuracy moved to 0.98 from 0.95 (YoloV8s)
- Multi camera accuracy moved to 0.9875 from 0.9808 (YoloV8s)



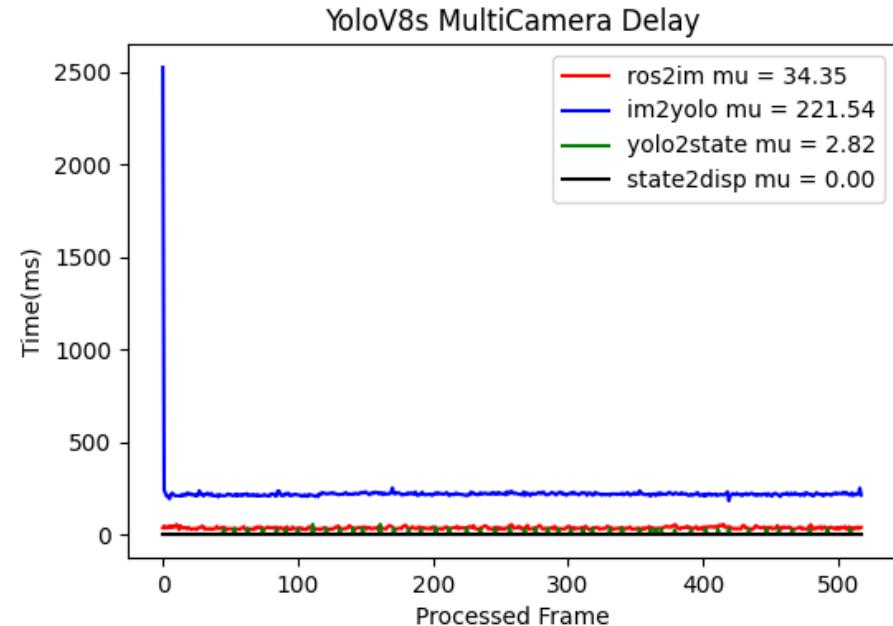
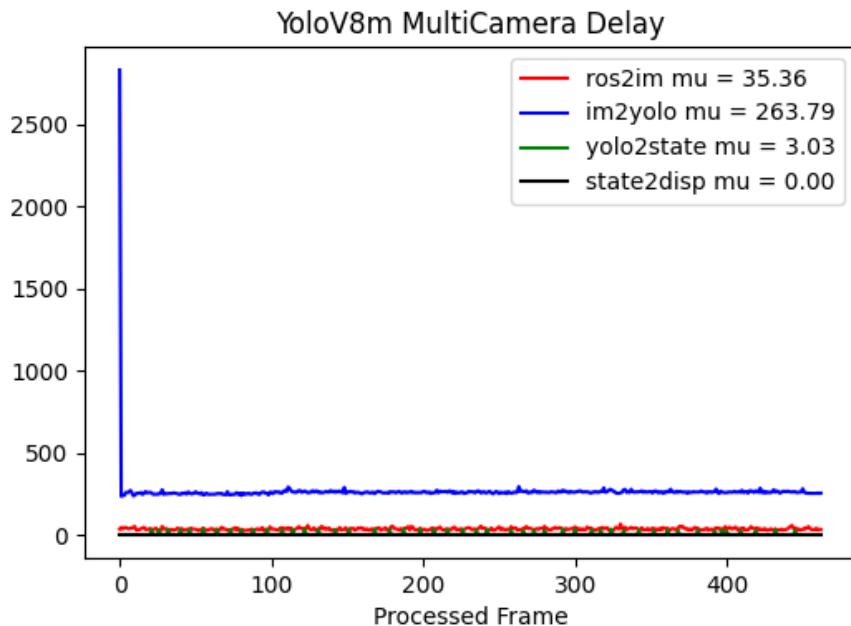
# Accuracy Results for YoloV8l (large)

- Single camera accuracy moved to 0.97 from 0.98 (YoloV8m)
- Multi camera accuracy moved to 0.9754 from 0.9875 (YoloV8m)
- Large model performs worse since we don't have enough data



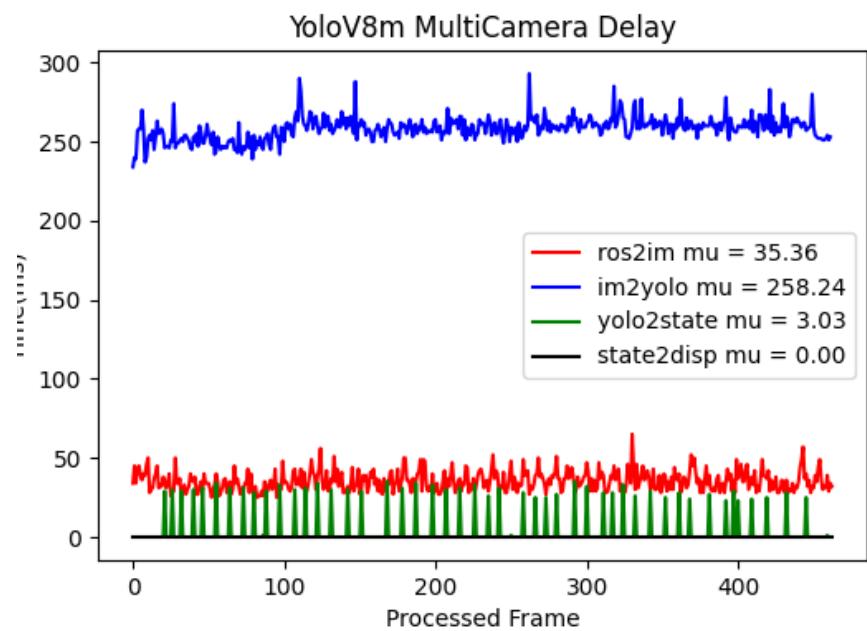
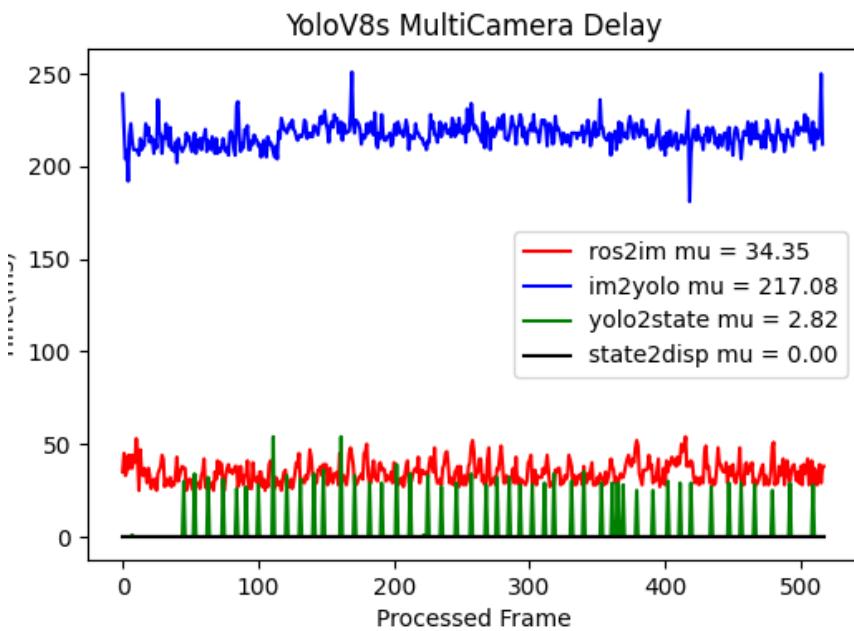
# Multi-view Camera: does it pay off?

- It brings a marginal improvement over accuracy.
- However, we need to detect corners at the beginning two times
- Also with initialization, first delay is higher.



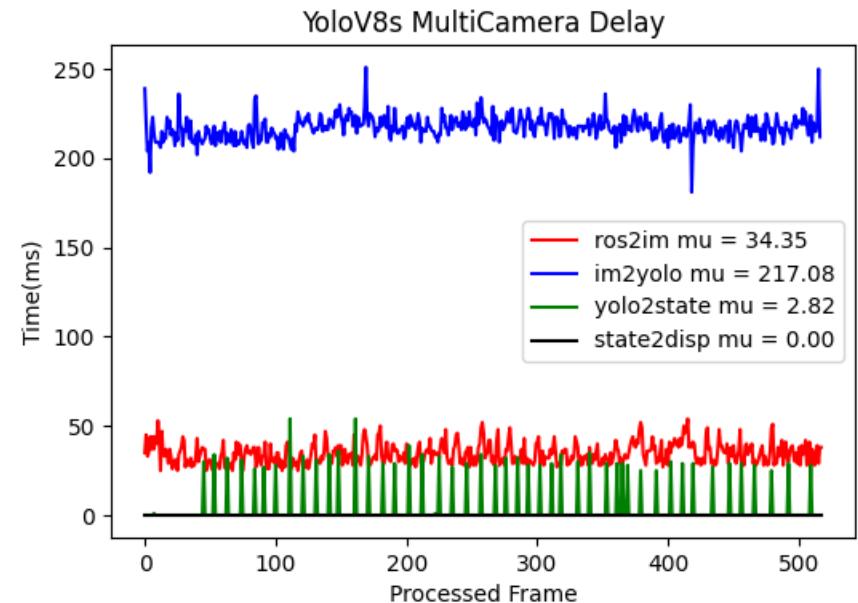
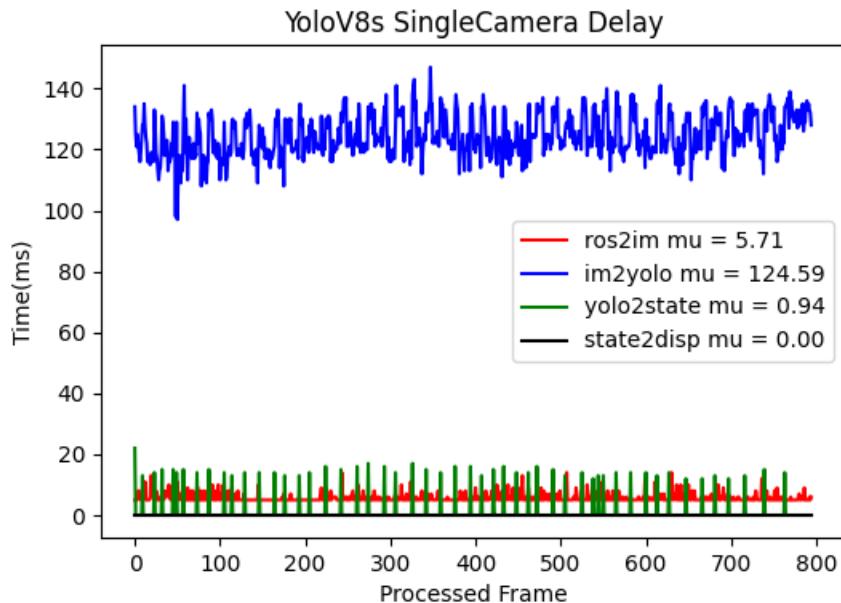
# Multi-view Camera: stable after first frame

- After initialization, it processes the frames in comparable times
- Since medium model has more parameter, it takes more time to compute prediction than that of the small model



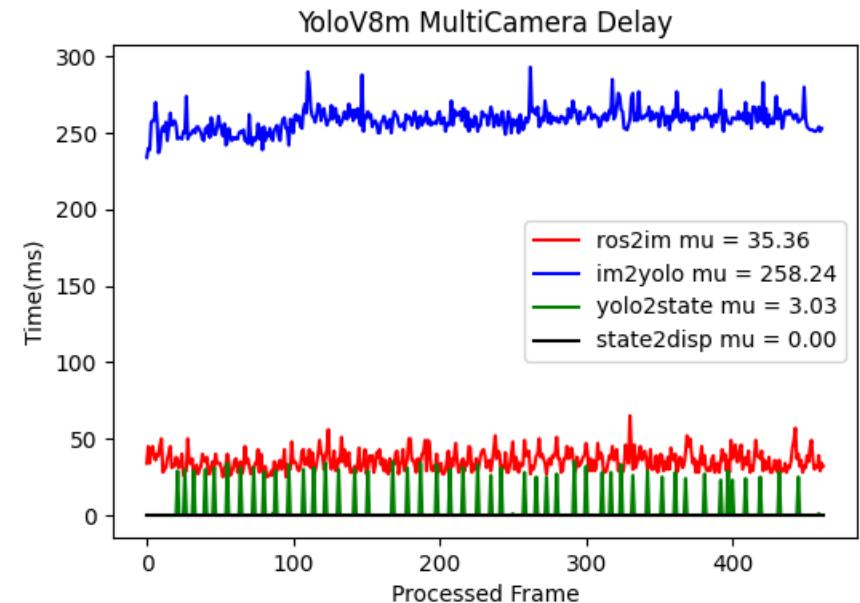
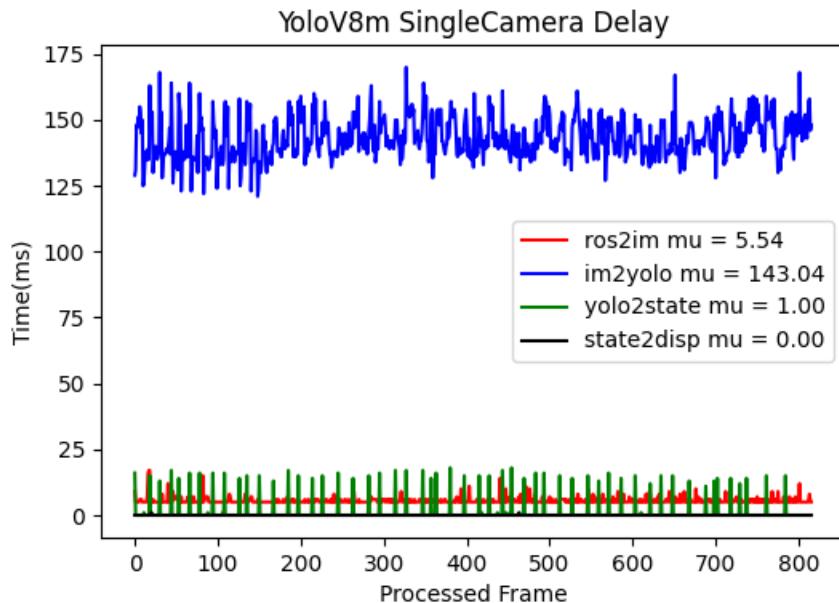
# Delay: Multi vs. Single-Camera

- Second camera is sequentially processed with the first one
- That causes the delay to almost double for every node
- Timings are given for small model

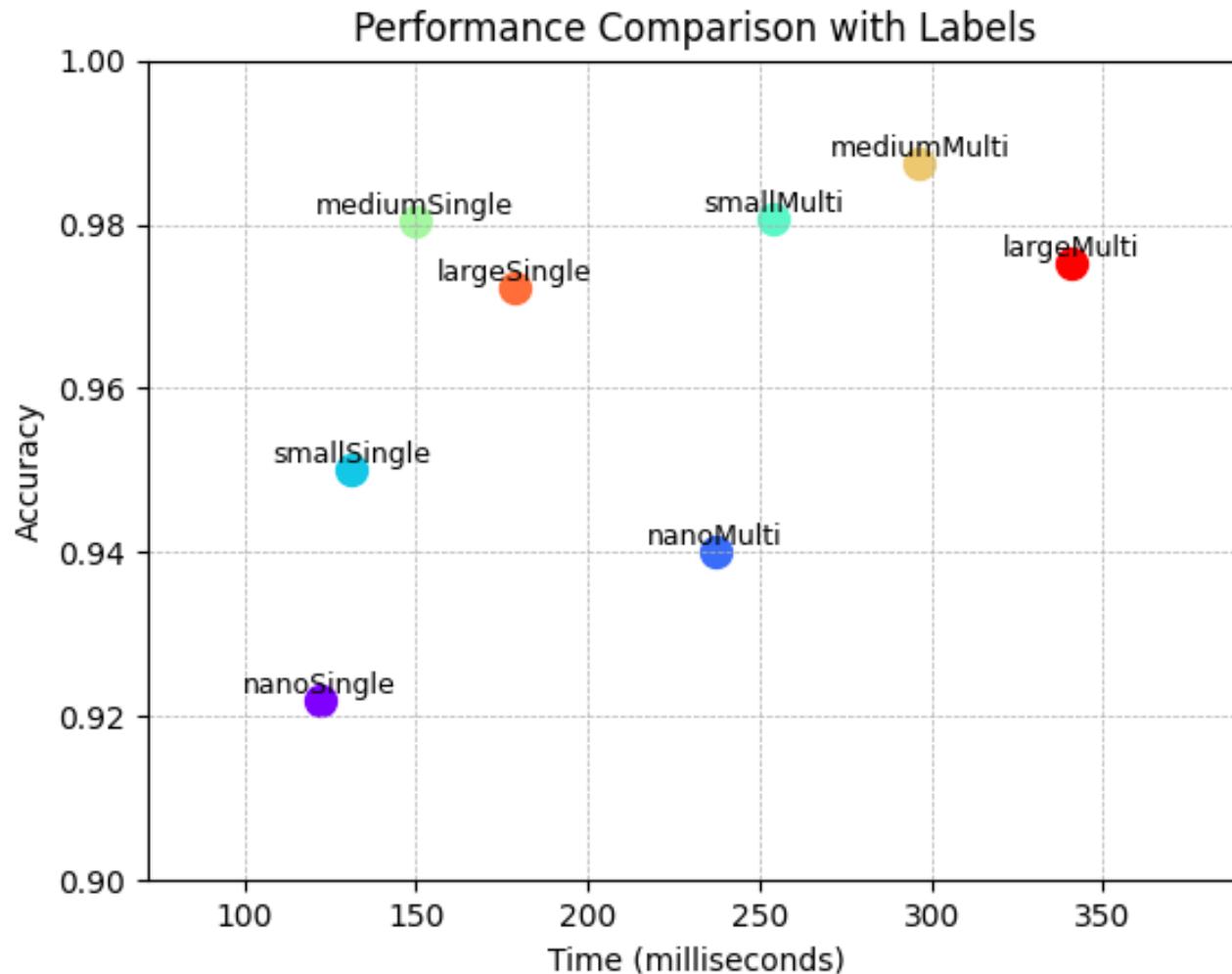


# Delay: Multi vs. Single-Camera

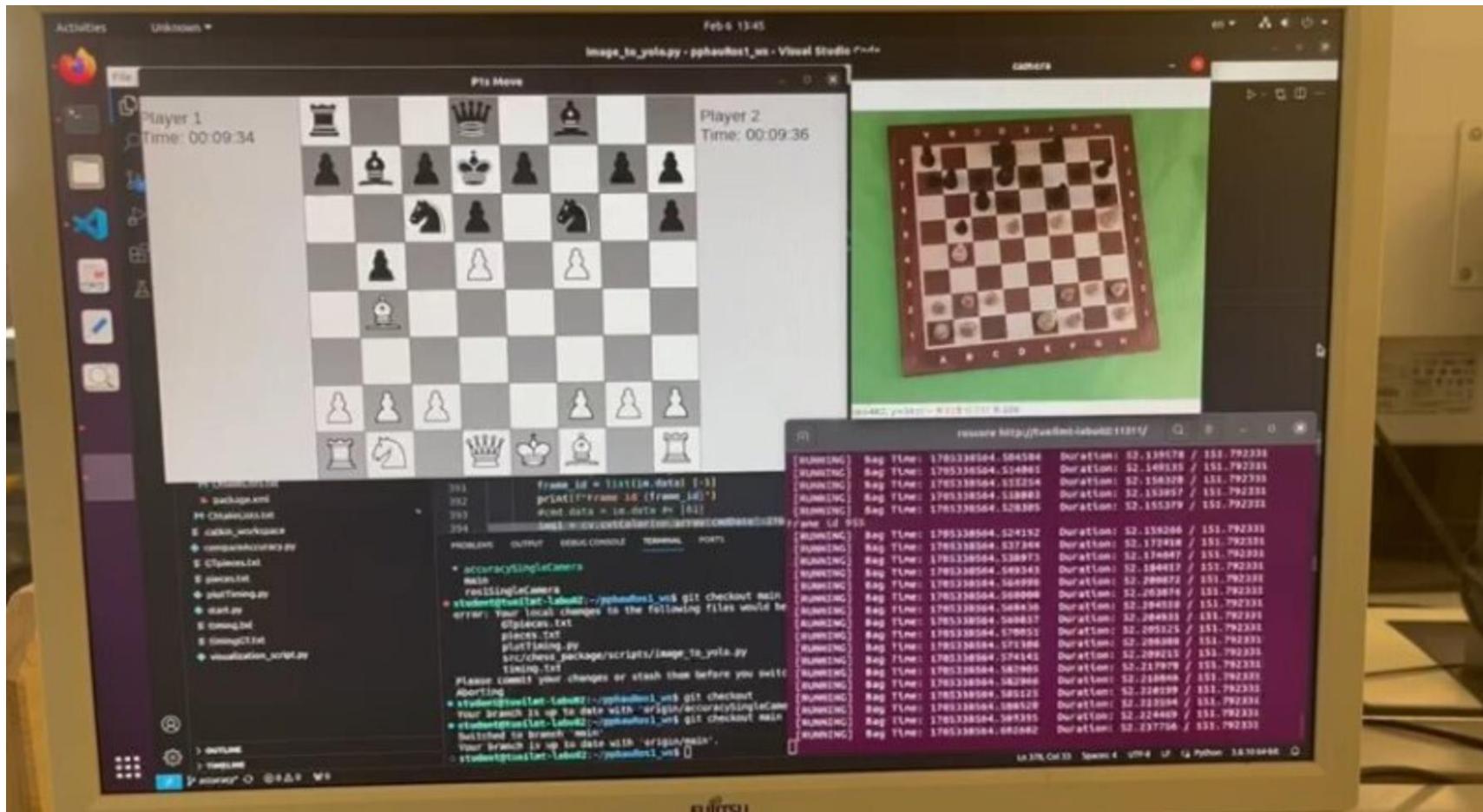
- Second camera is sequentially processed with the first one
- That causes the delay to almost double for every node
- Timings are given for medium model (increased w.r.t. to small)



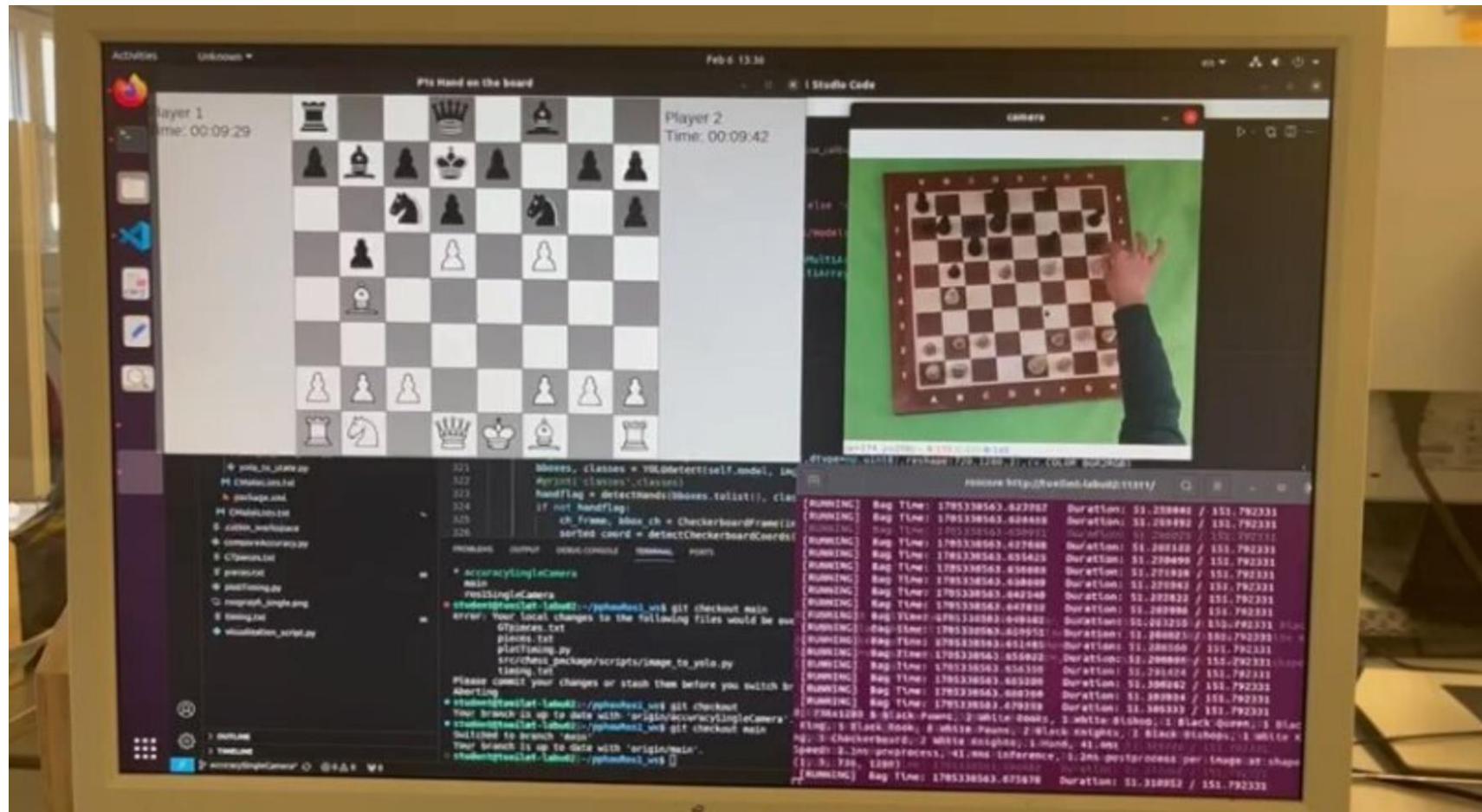
# The trade-off of speed and accuracy



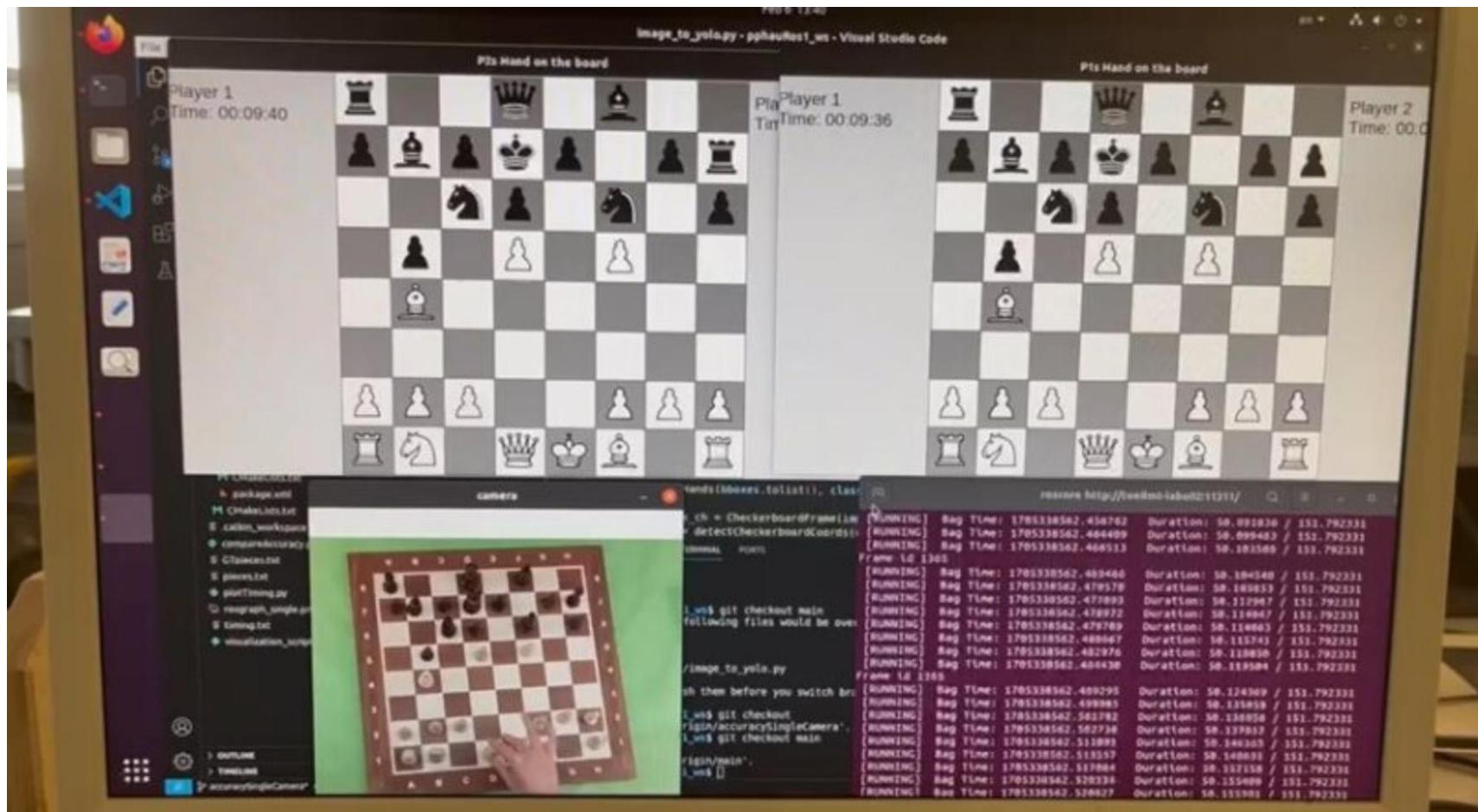
# DEMO: YoloV8m, MultiCamera



# DEMO: YoloV8m, SingleCamera

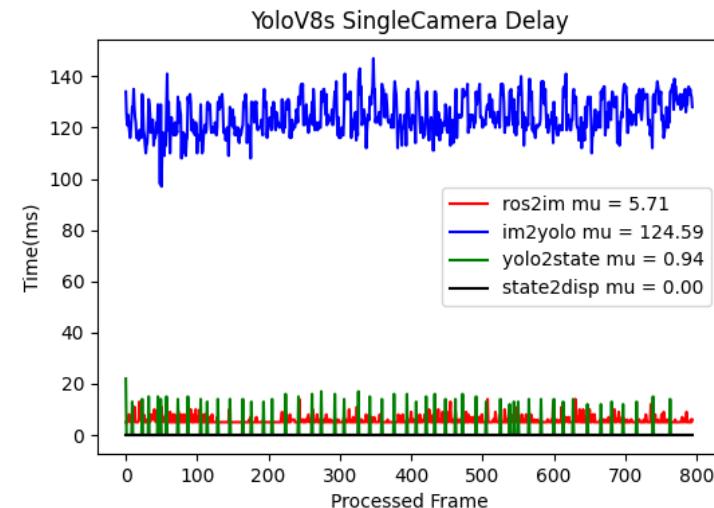
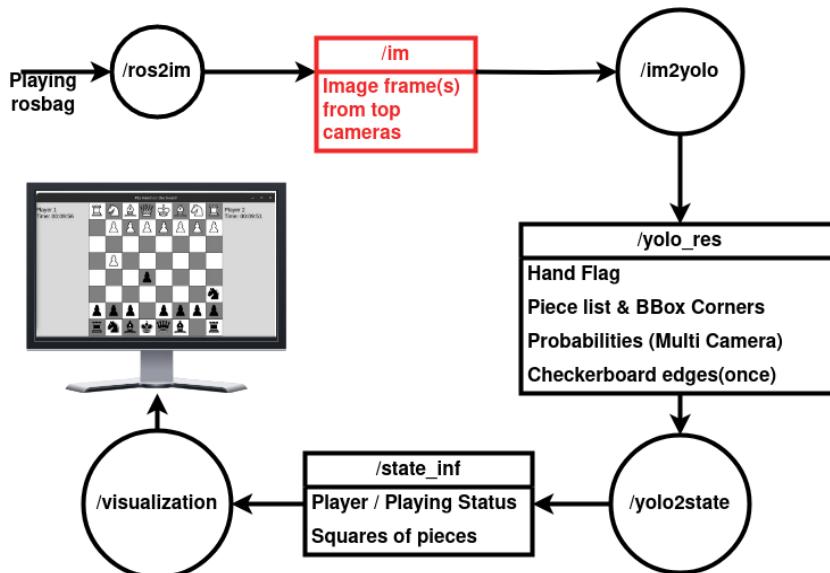


# DEMO: Single Camera with Ground Truth



# FUTURE WORK

- One node creates too much delay comparatively
- Buffer of /im topic can become unstabilized
  - Suggestion: Re-arrange the node-topic pipeline for app. eq. Delays
  - Challenge: Each code snippet should be taken into account





# QUESTIONS