VISUAL ODOMETRY & AUTO CAR

AGBARYA WEAAM
EGBARIA MOBEEN
EGBARYA ṪHAWAB

MVO&AUTO

INTRODUCTION

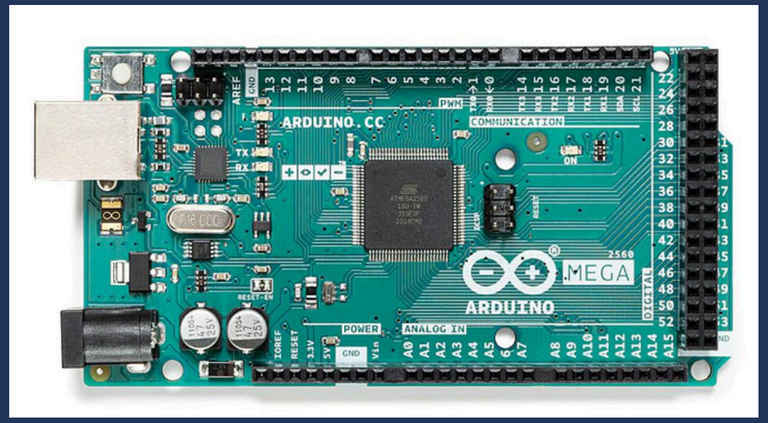
- Our final project is a combination of two projects under robotics: Visual Odometry and Autonomous Car.
- The goal of the Visual Odometry project is to accurately estimate the motion of a robot using visual information from a camera mounted on the robot.
- while the Autonomous Car project aims to develop a robot that can detect road lanes and autonomously navigate through them, as well as detect pedestrians and road signs using machine learning.

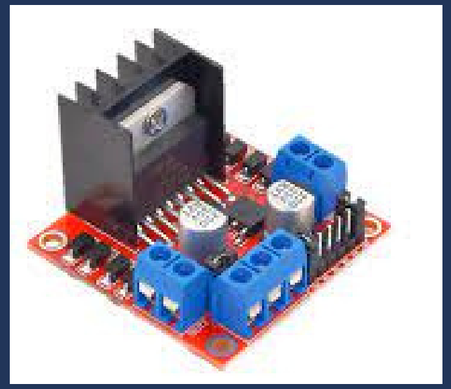
ARDUINO

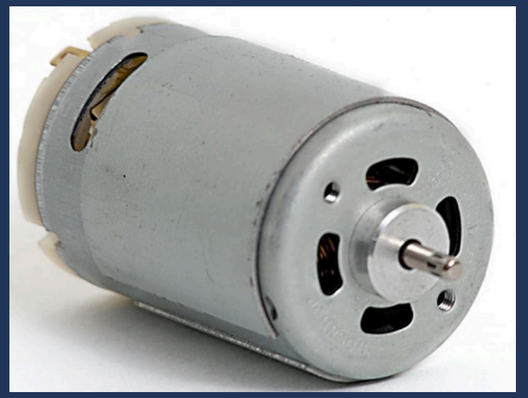
we used :

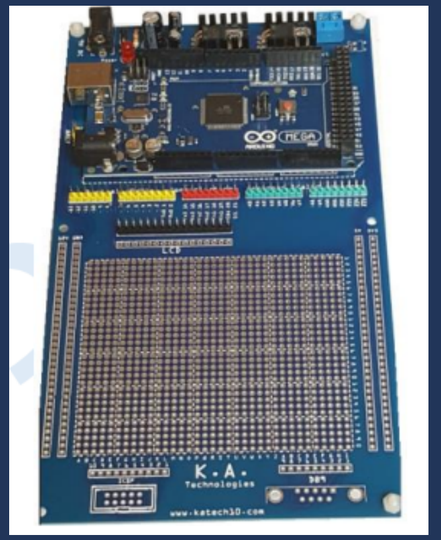
- Arduino mega
- Three wheels
- 2 DC Engines
- 12v Battery
- L298N
- Bluetooth Sensor HC06
- Phone holder

Robot Components

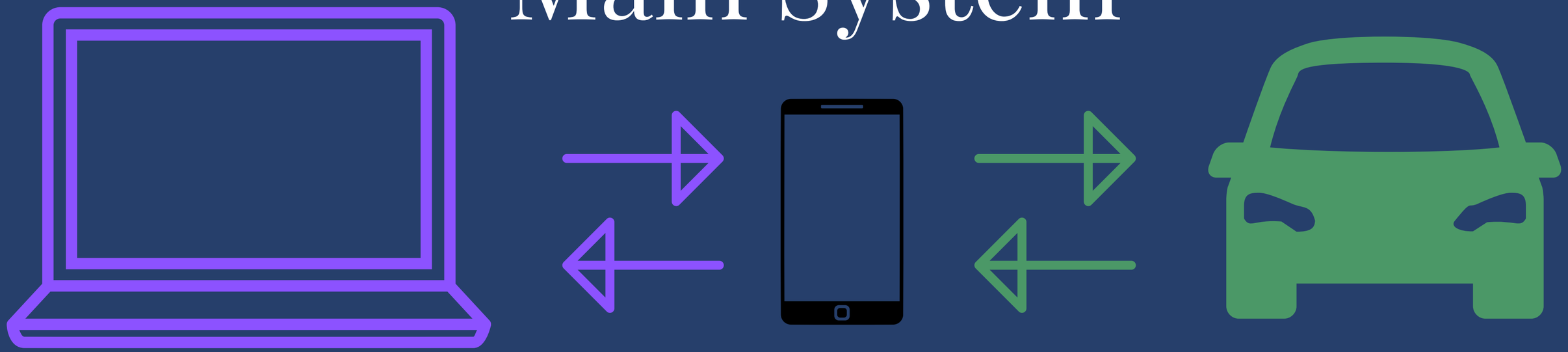








Main System



Visual Odometry :

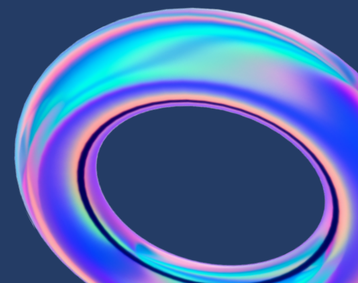
- every ~200ms (5 frames) the phone sends video data
- the laptop handels the computation and analysis
- output displayed on website

Autonomuos Car :

- every ~200ms (6 frames) the phone sends video data
- the laptop handels the computation and analysis
- sends back the output command to the arduino

VISUAL ODOMETRY

Visual Odometry is a technique used in robotics to estimate the motion of a robot using visual information from a camera mounted on the robot. It is often used in autonomous vehicles to navigate and understand the environment.



VISUAL ODOMETRY

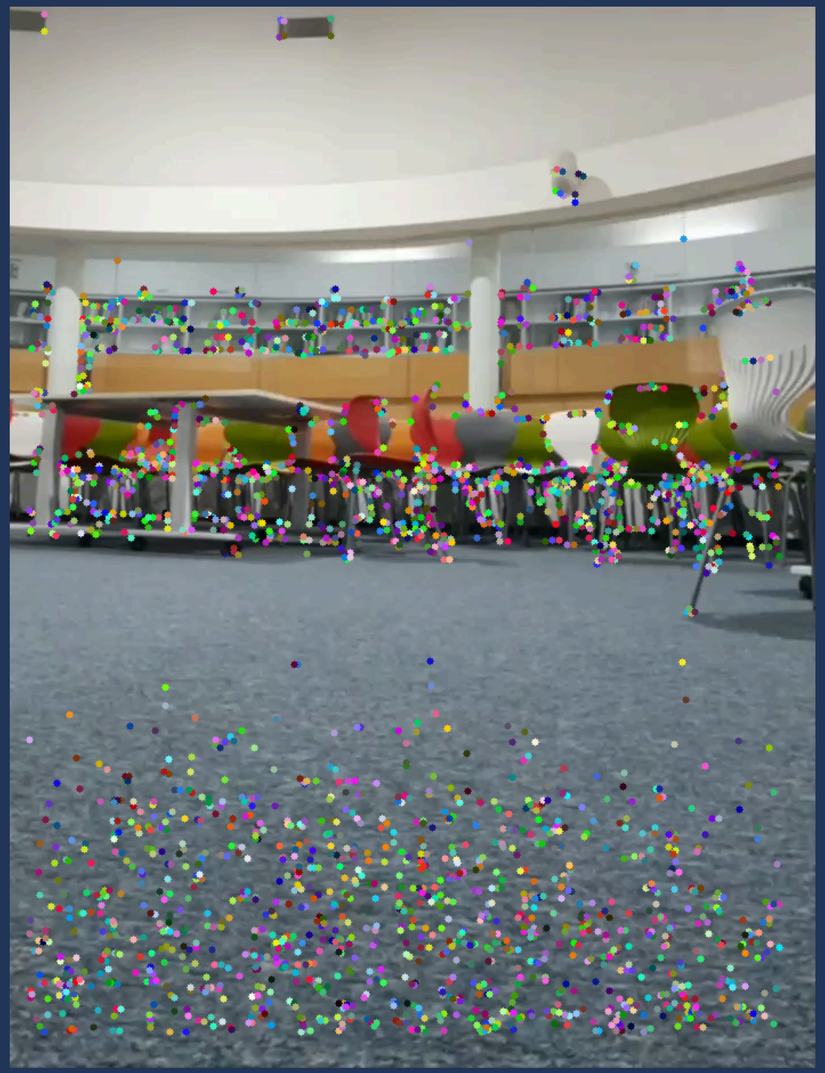
FEATURE DETECTION AND MATCHING

The first step in Visual Odometry is to detect and match keypoints (features) in the current and previous frames

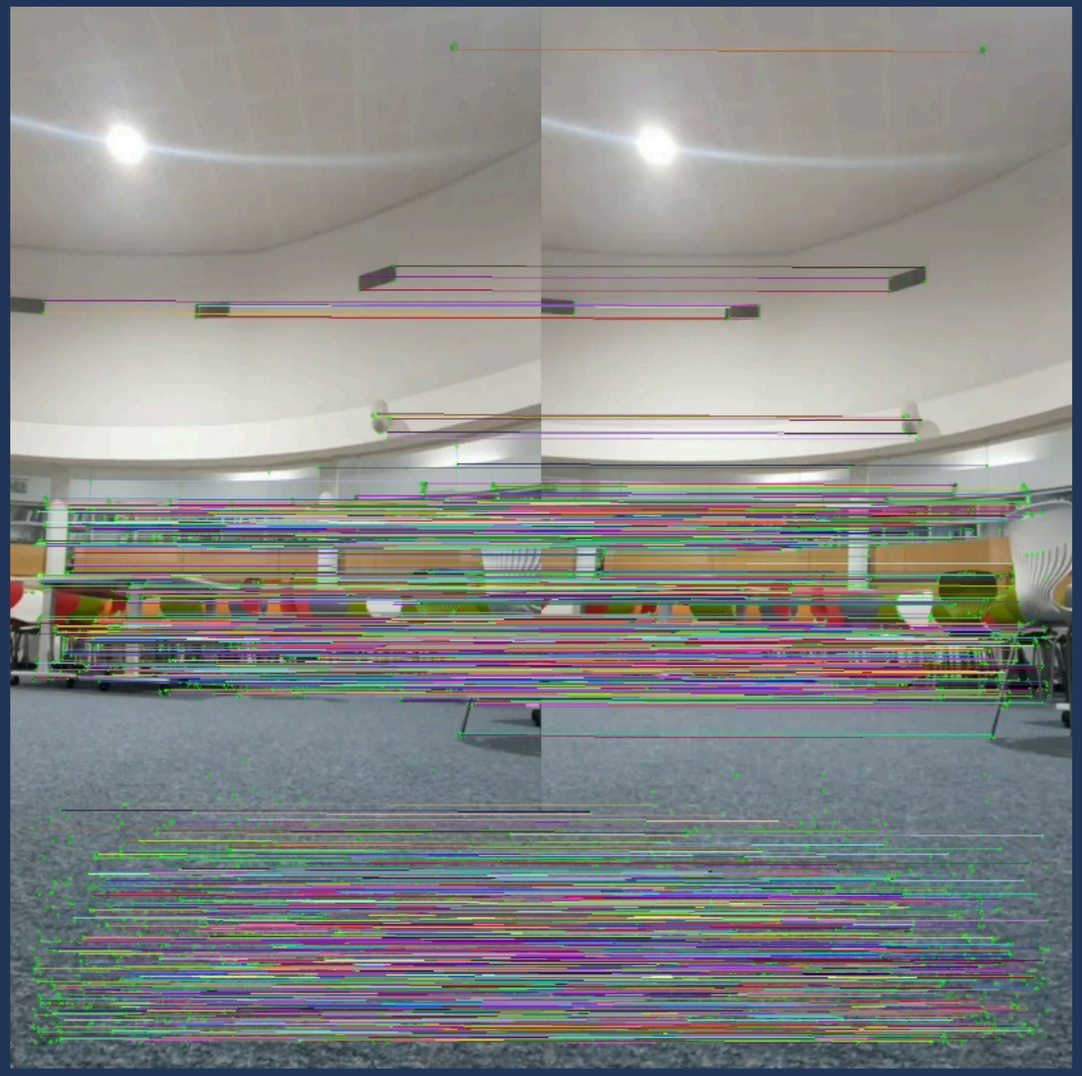
ESTIMATE RELATIVE CAMERA POSE

Once the keypoints have been matched, the next step is to estimate the relative camera pose using the matched keypoints

Features detection



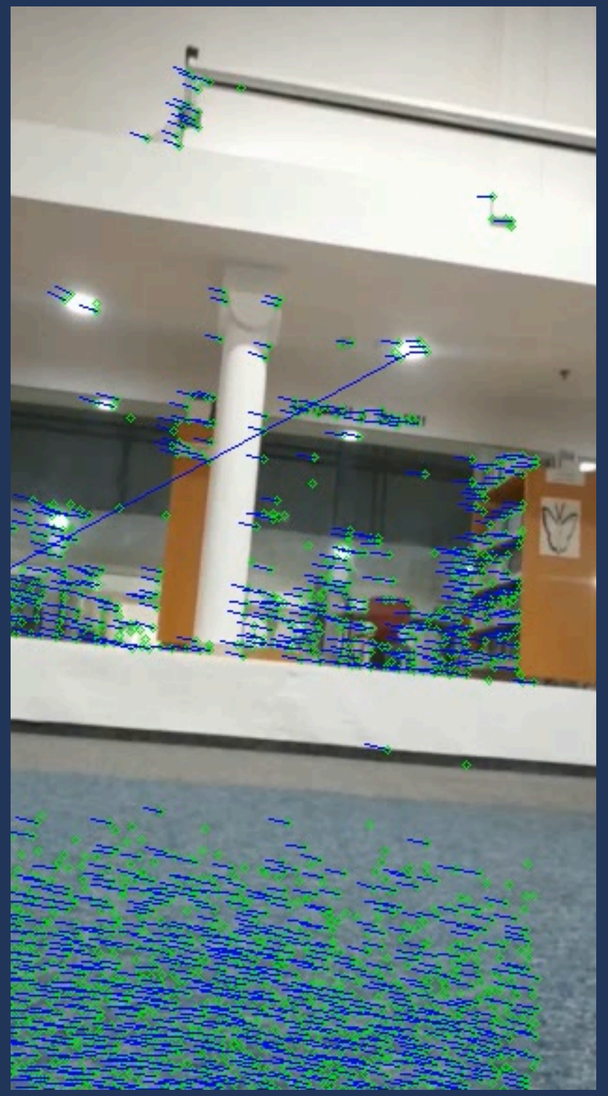
Matching

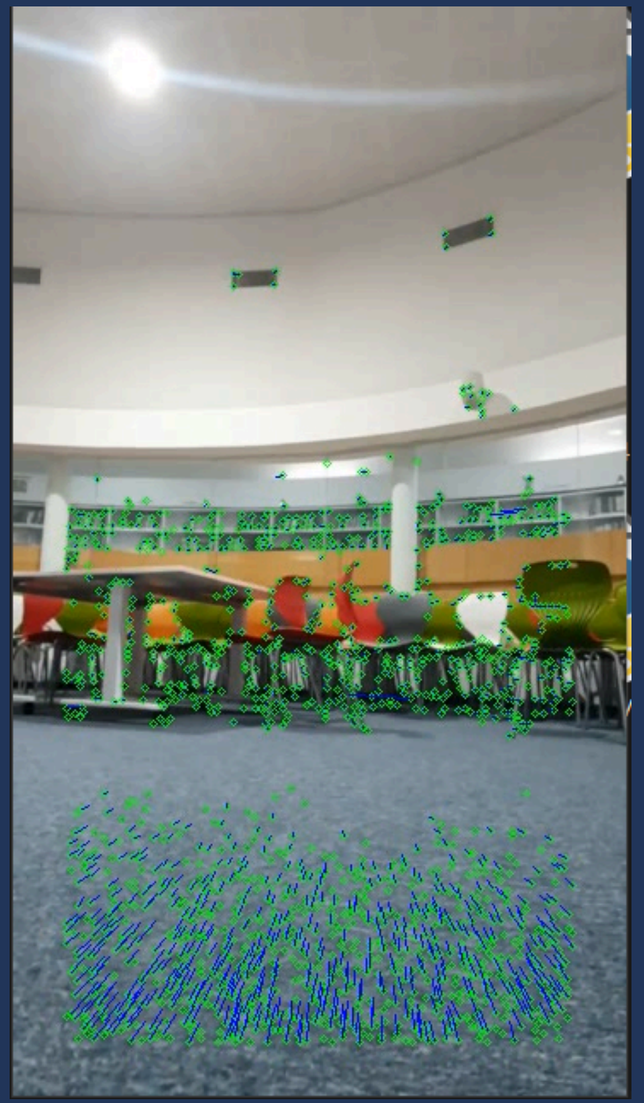


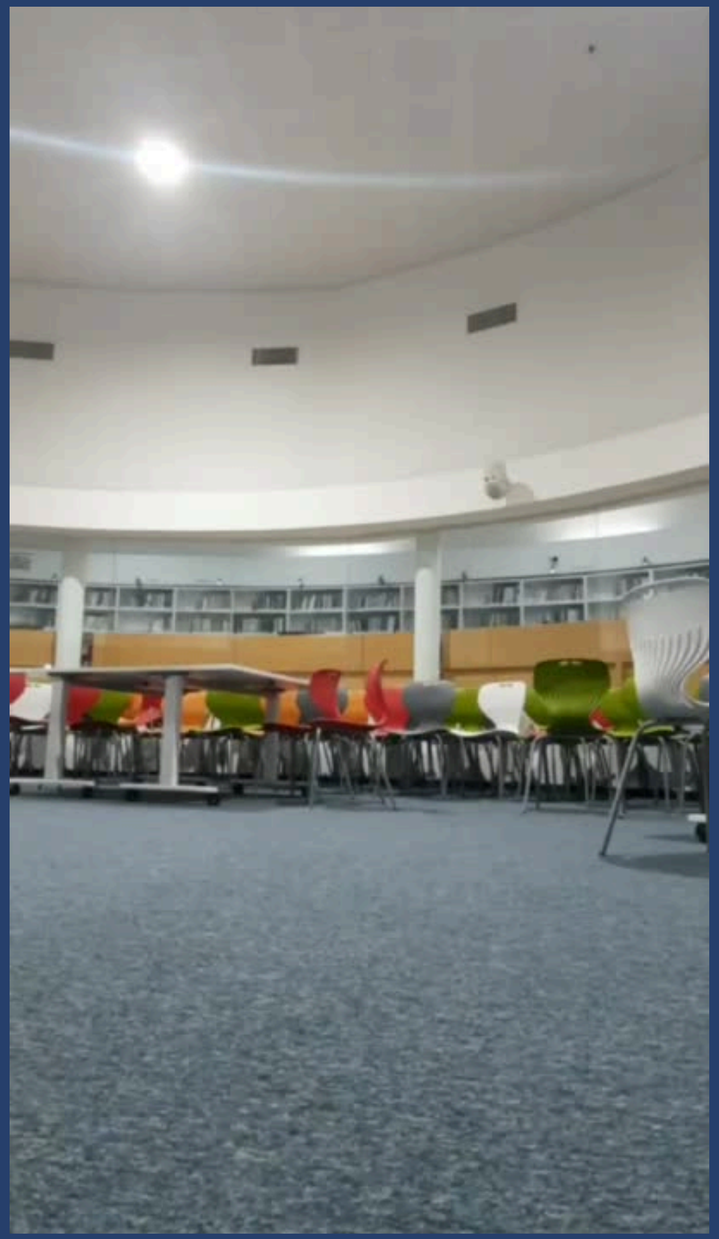
- **Estimating Pose**

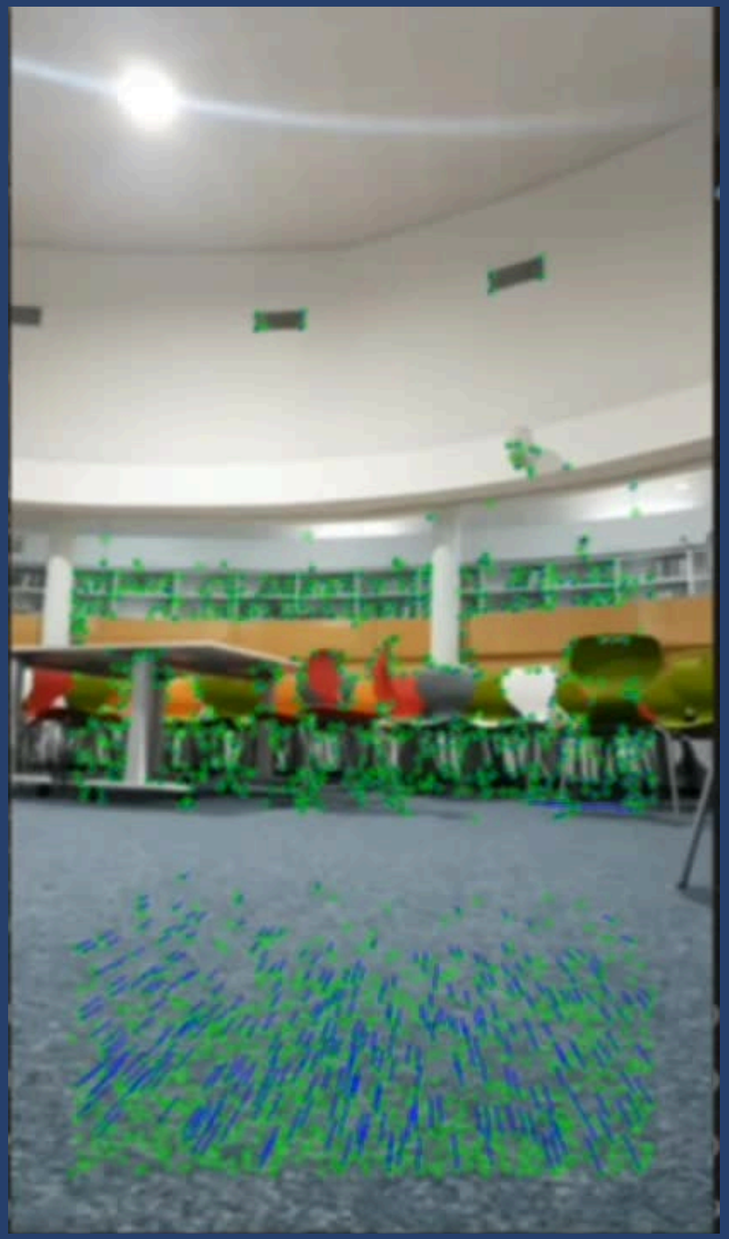
Going to a certain direction

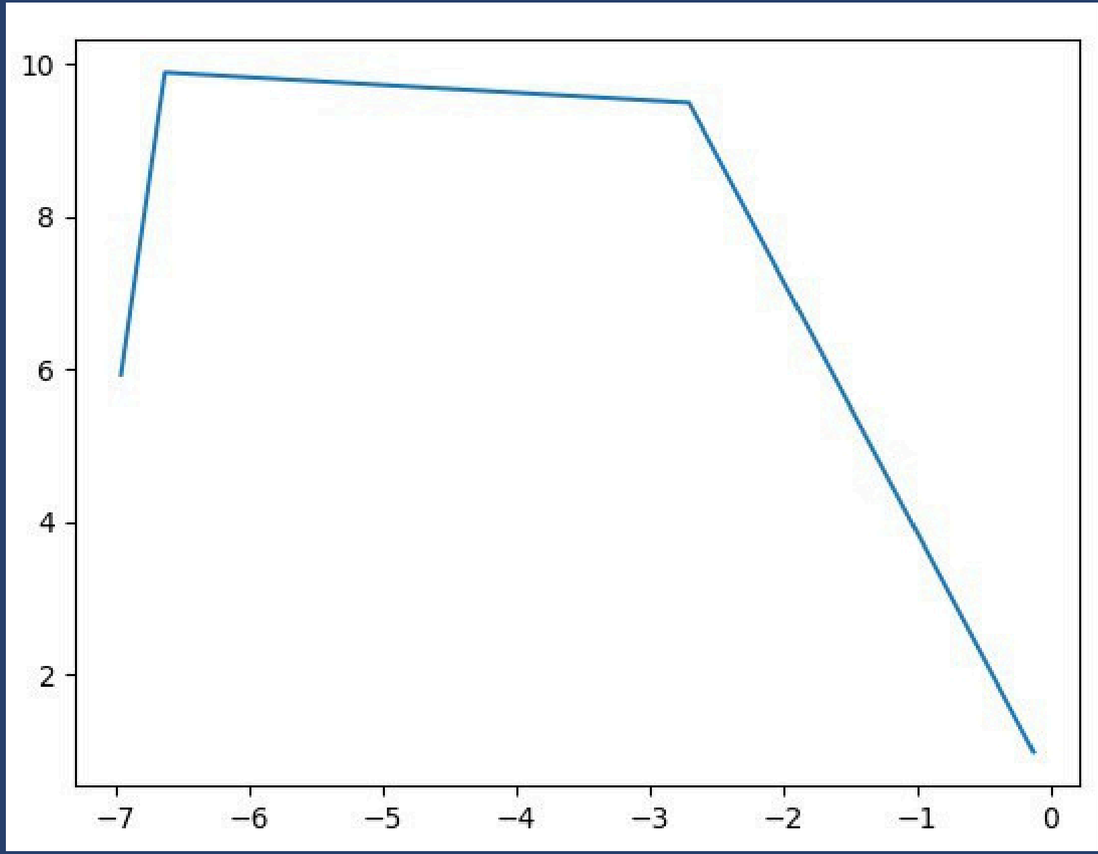
Features will be going opposite of the direction we are going to.
So going forward we will see that the features are going inwards to us/backwards, And so on for other directions.

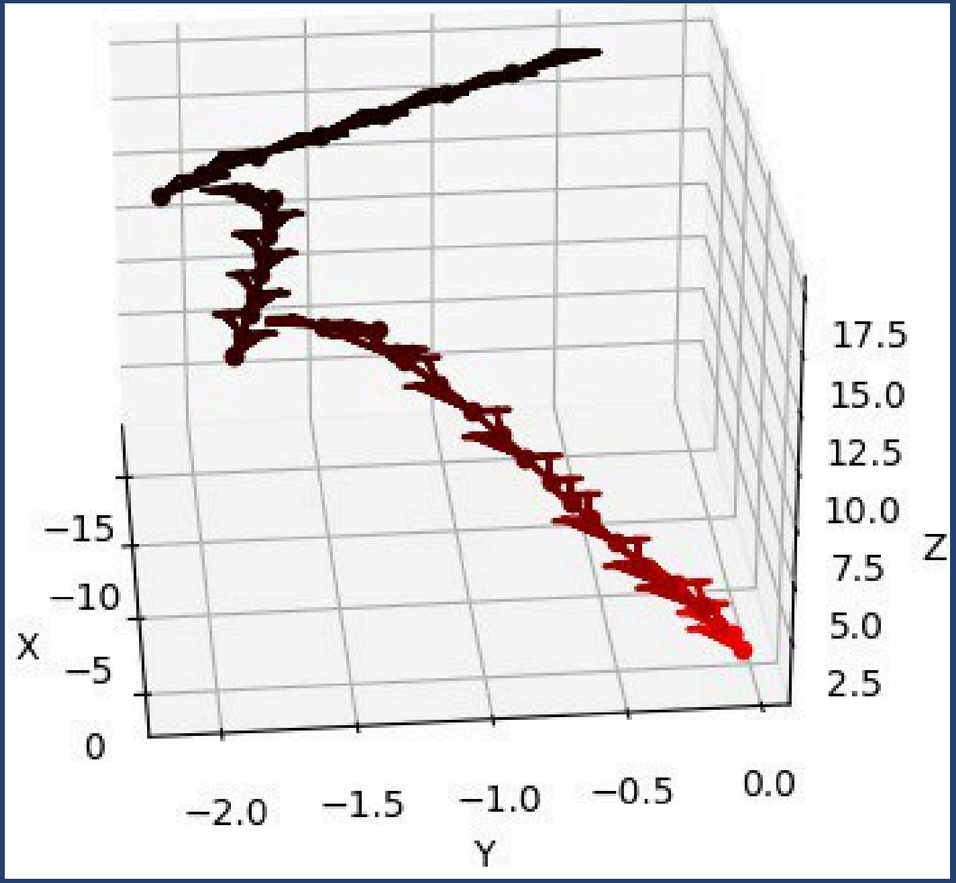


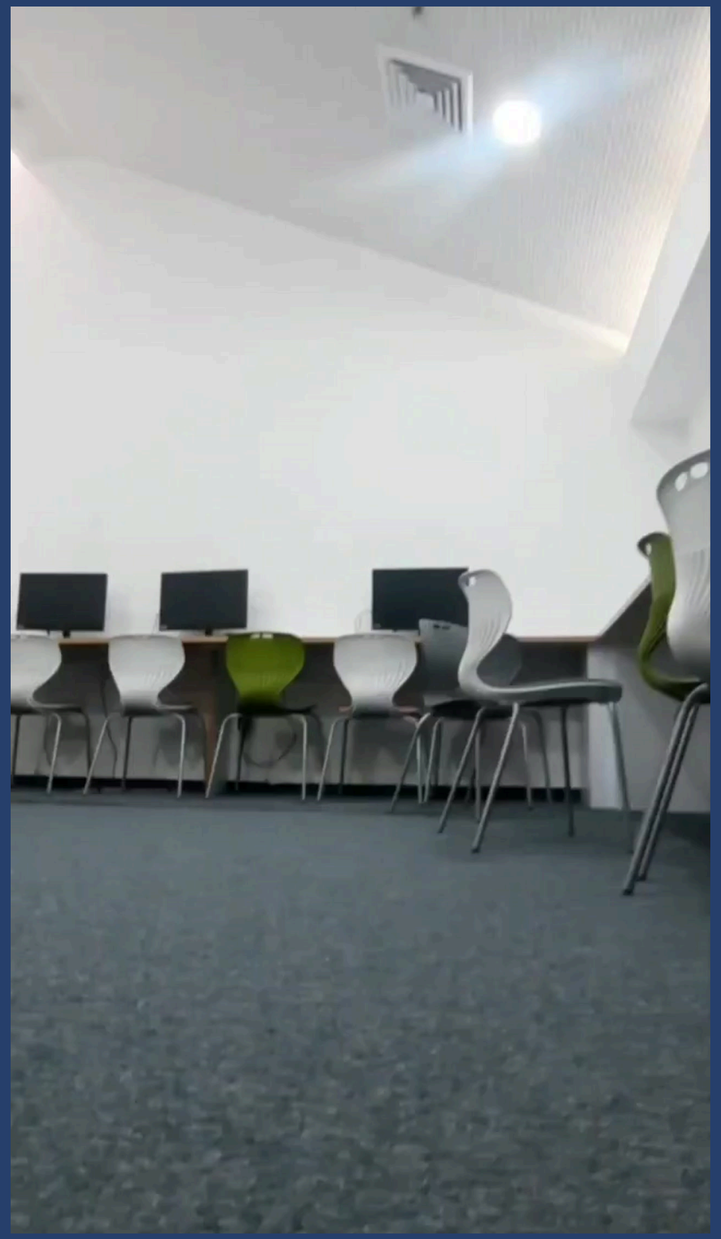


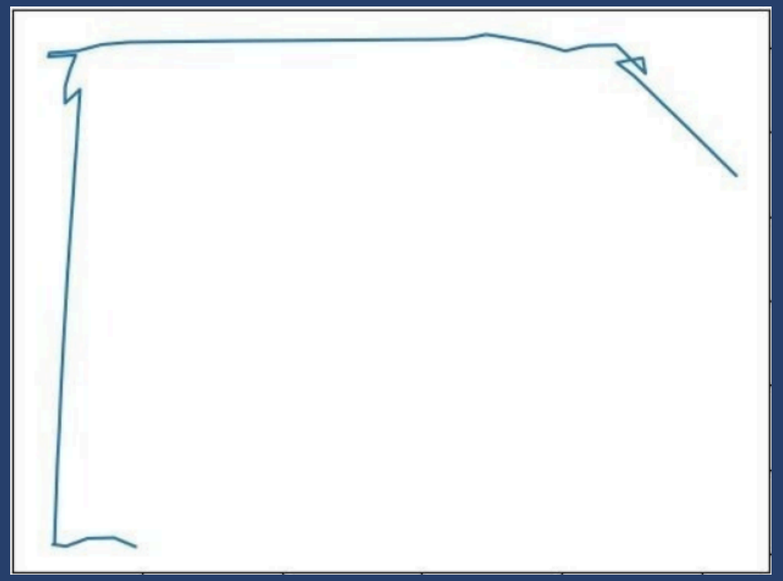


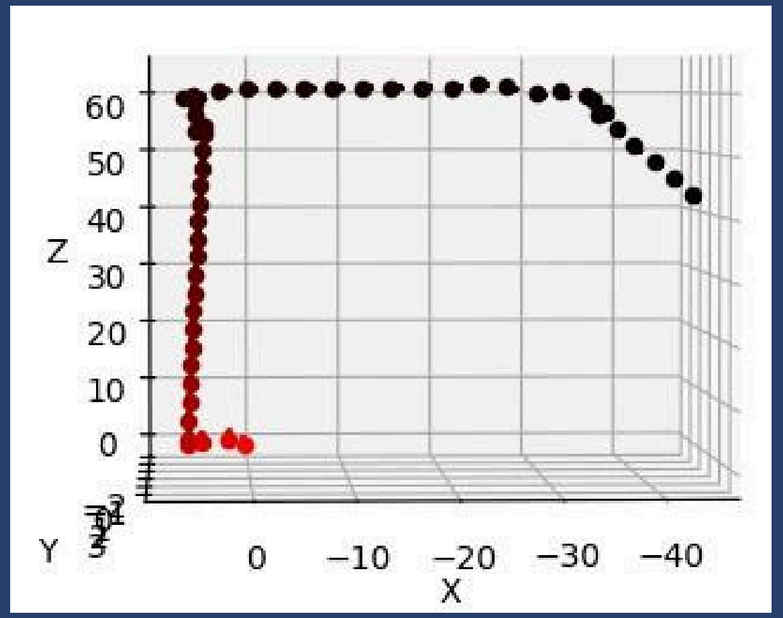


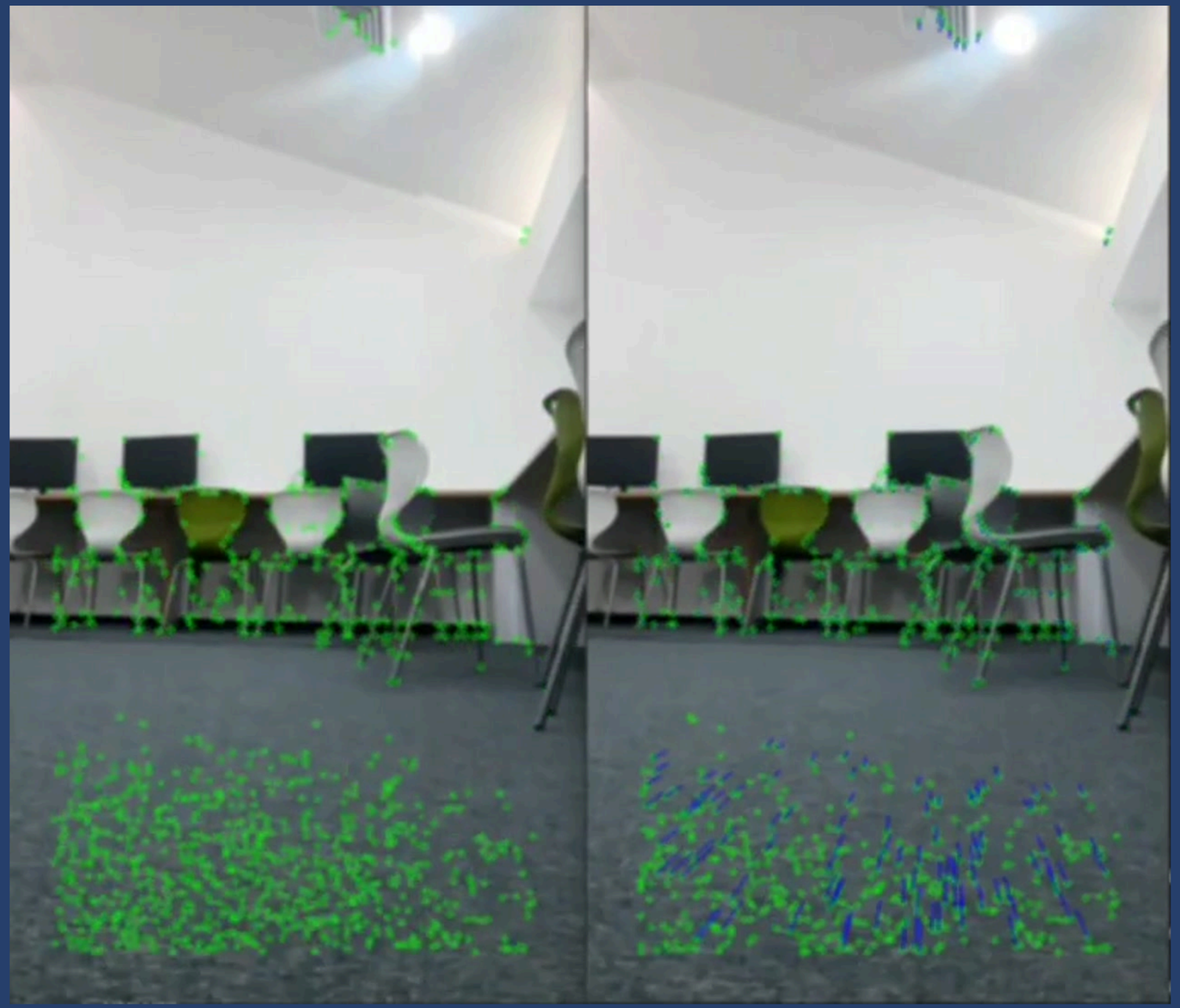

10
8
6
4
2
−7
−6
−5
−4
−3
−2
−1
0

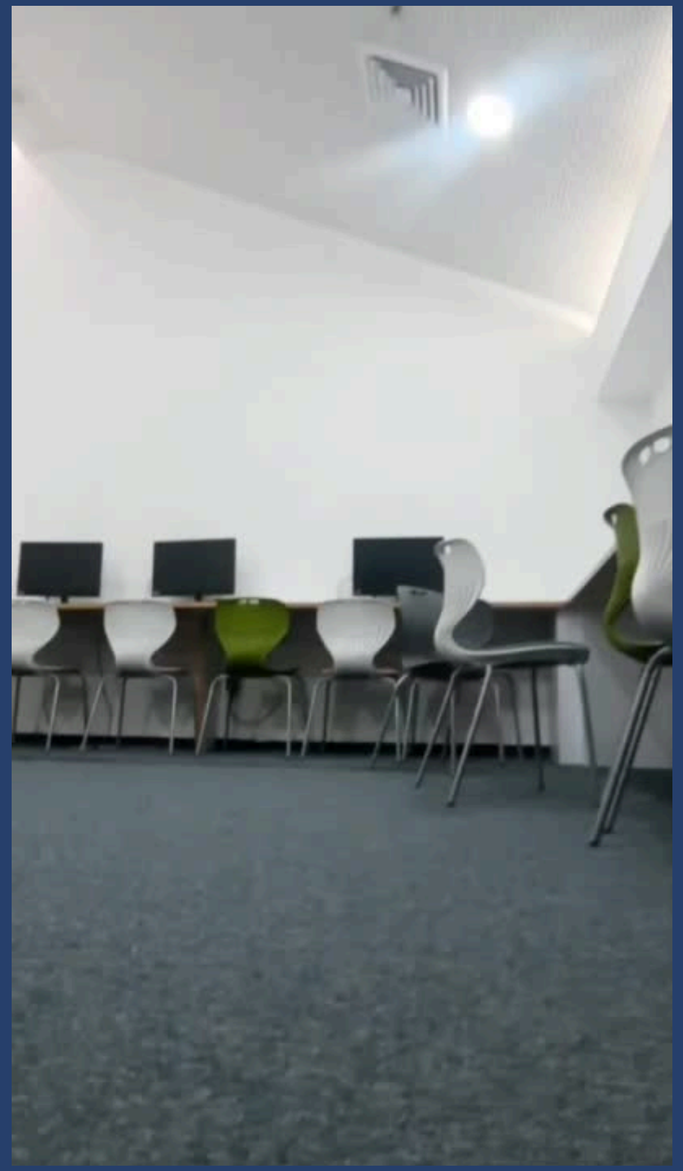

17.5
15.0
12.5
10.0
7.5
5.0
2.5
Z
−15
−10
−5
0
X
−2.0
−1.5
−1.0
−0.5
0.0
Y

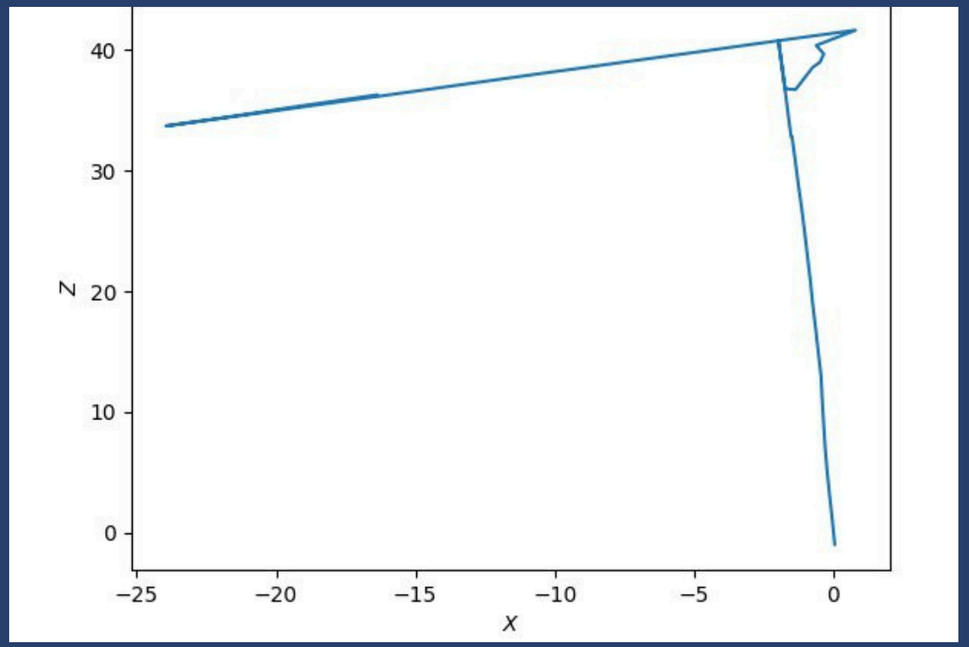


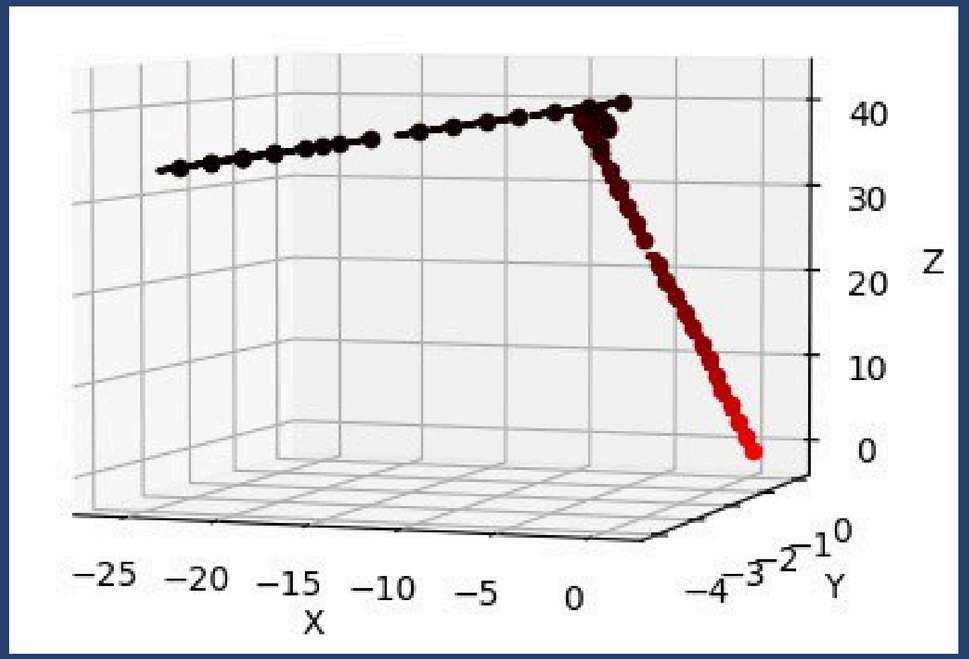



60
50
40
Z
30
20
10
0
Y
0
−10
−20
−30
−40
X






40
30
20
10
0
Z
−25
−20
−15
−10
−5
0
X


40
30
20
10
0
Z
−25
−20
−15
−10
−5
0
X
−4
−3
−2
−1
0
Y

Challenges We Encountered

There are several limitations to our system:

- Quality and resolution of the input images
- Presence of noise or other factors can interfere with the feature detection and matching process.
- Sensitivity to changes in lighting or other environmental conditions.
- Changing phone/camera requires changing parameters.

AUTO - NOMOUS CAR

Driving through the road

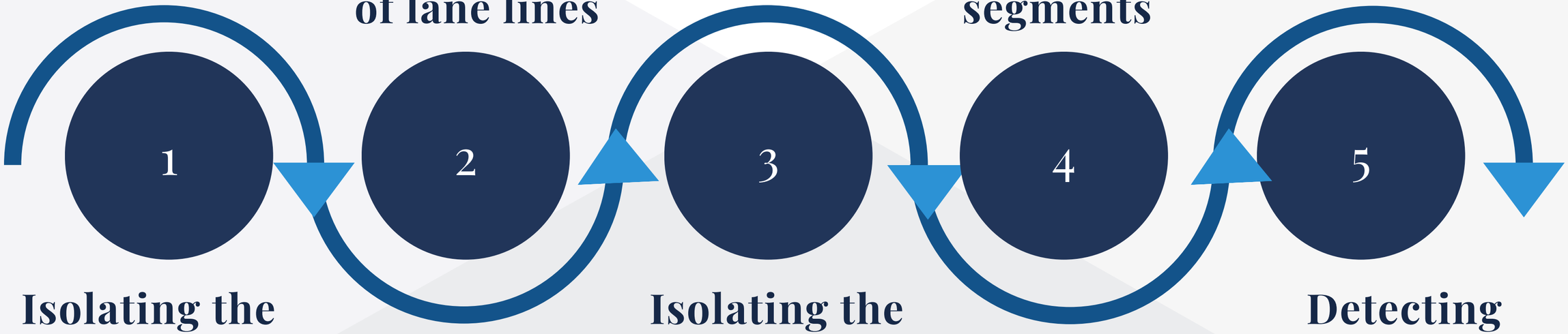
Autonomous car needs to drive autonomously through street (In our case blue tape)

Recognizing Traffic signs

The car has to recognize the pedestrians and traffic signs that we put on,
by training a custom model.

WALKING BETWEEN THE LINES

5-Step Process



Isolating the color of the lanes

Detecting Edges of lane lines

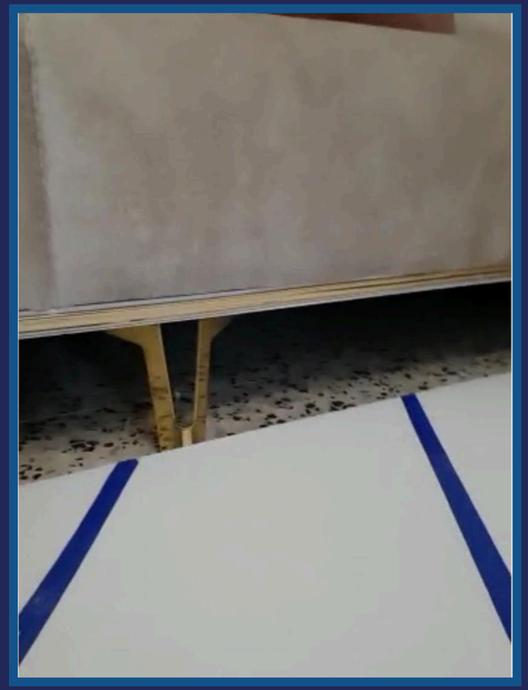
Isolating the region of interset

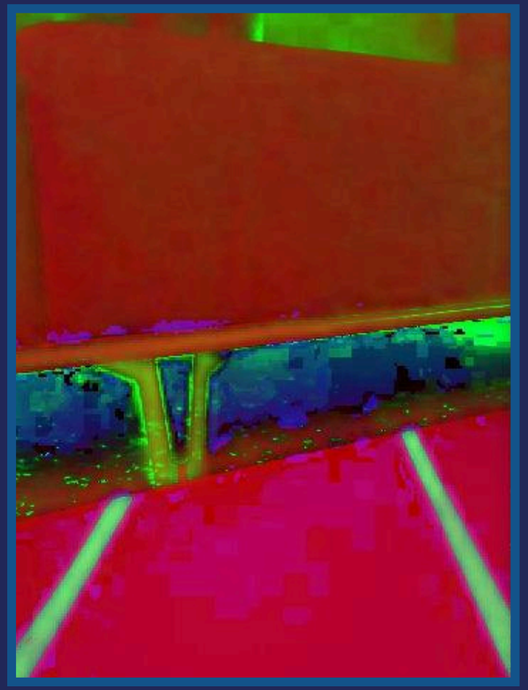
Detecting line segments

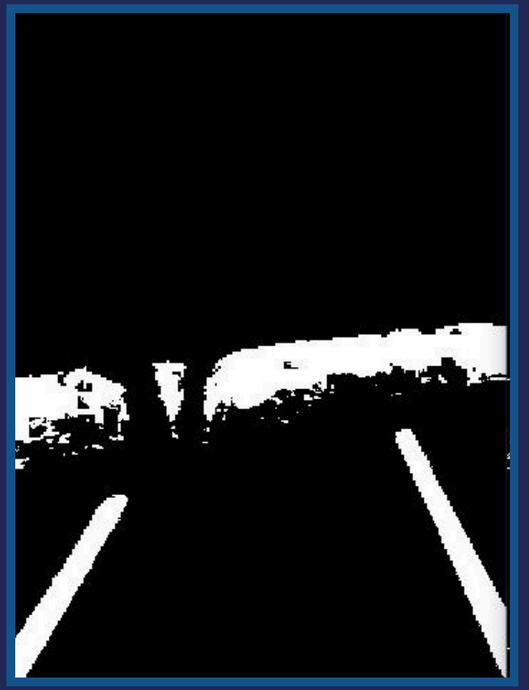
Detecting steering angel

Isolating the color of the lane

- We turned the color space used in the image which is RGB into HSV color space.
- And we lifted all the "blueish" colors from the image (by specifying a range of the color blue).

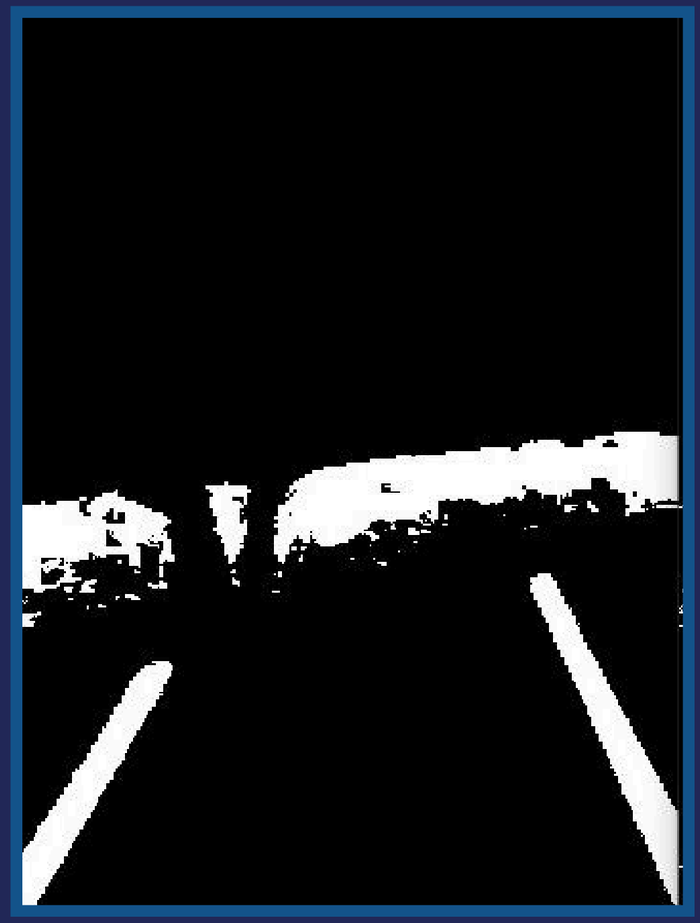


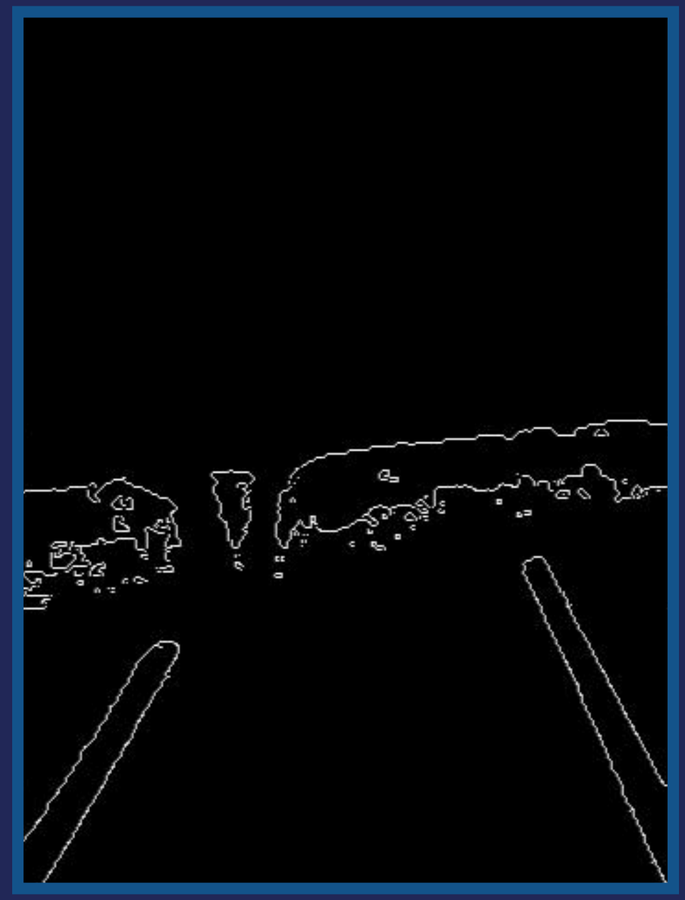




Detecting edges of lane lines

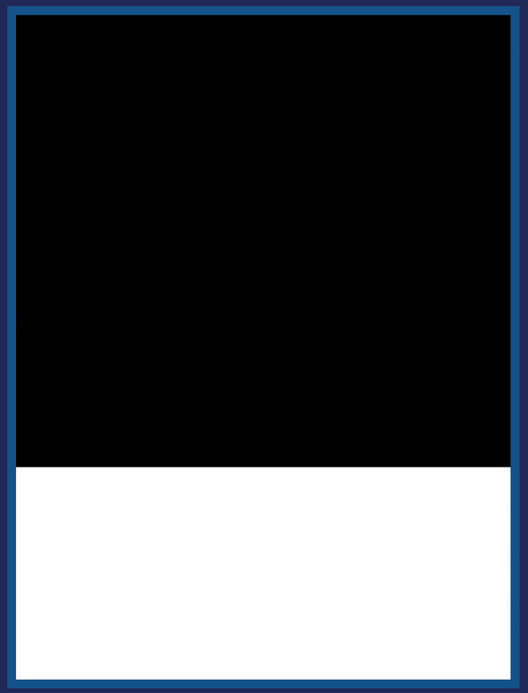
We extracted edges of all the blue areas by Canny function.

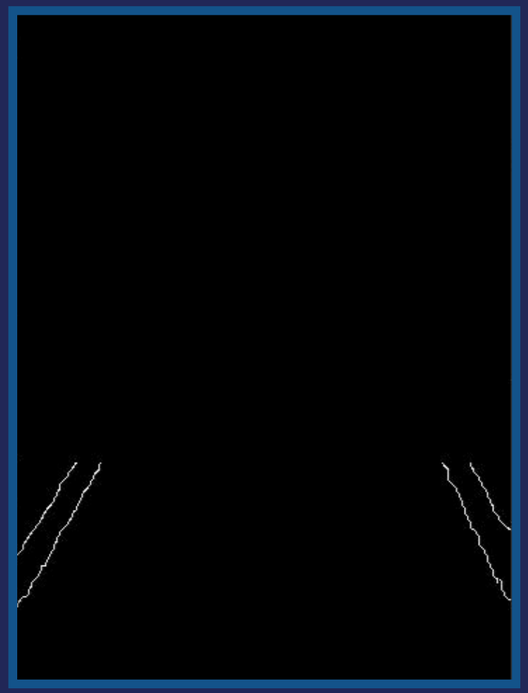




Isolating the region of interest

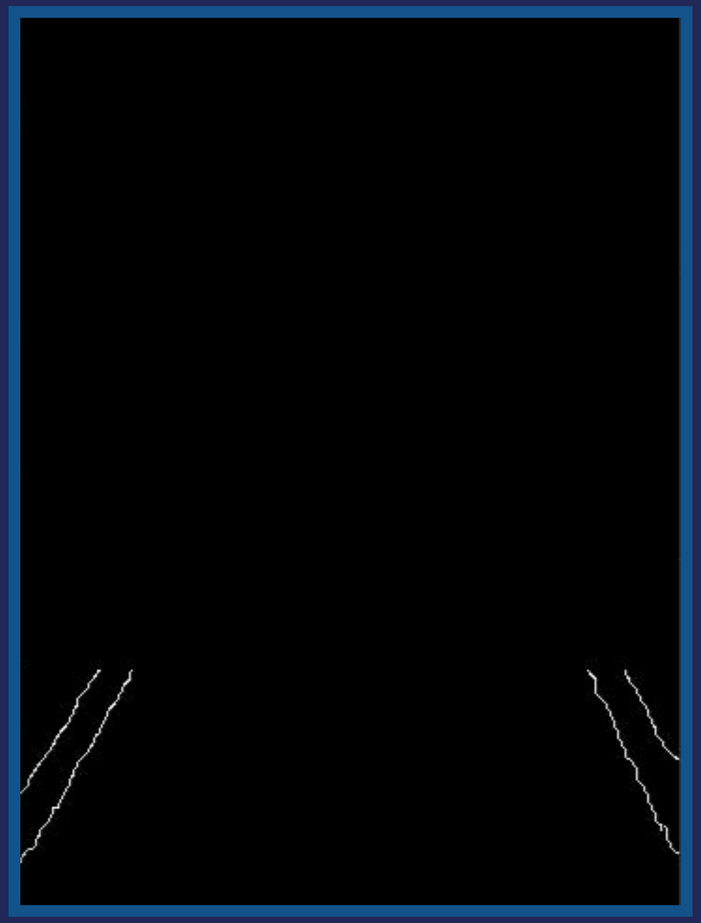
We detected quite a few blue areas that are NOT our lane lines.
So we simply crop out the top half of the image.

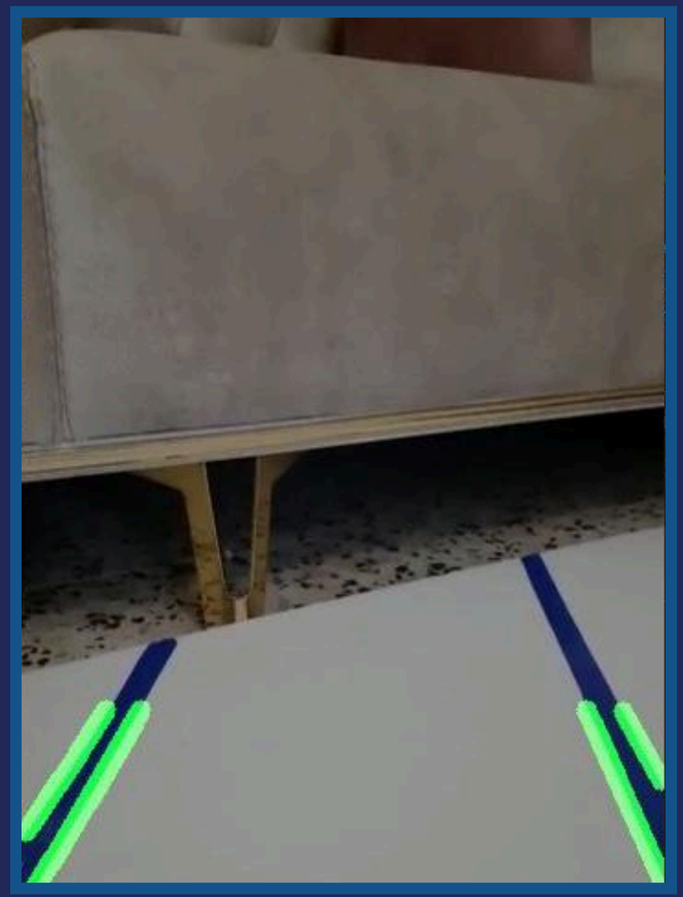




detecting line segments

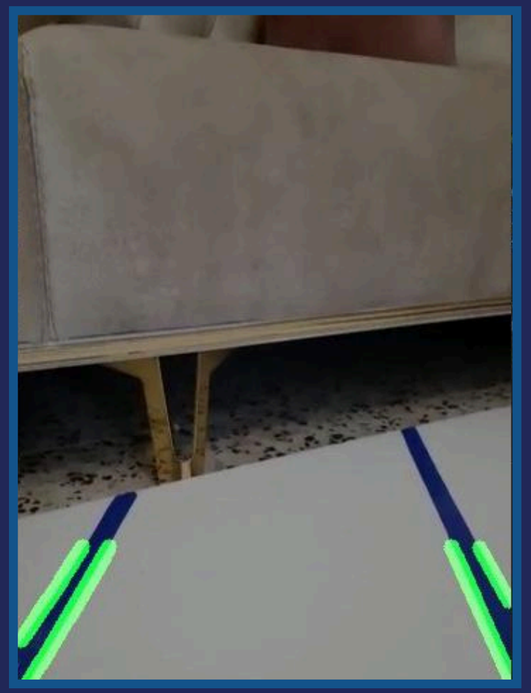
We extracted the coordinates of these lane lines from the white pixels.

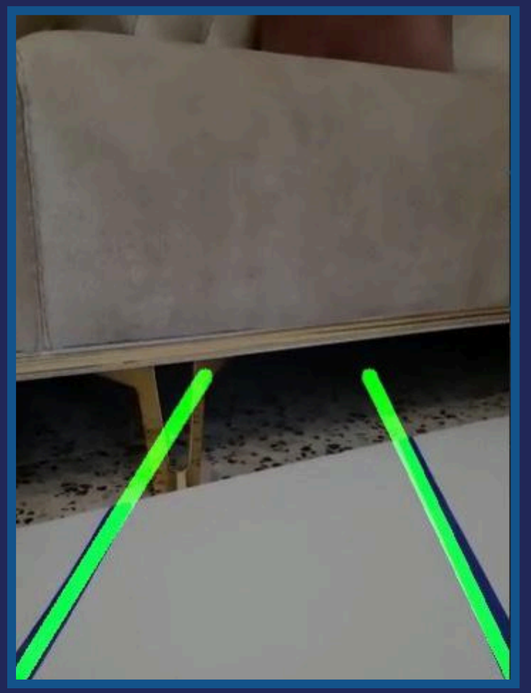




detecting steering angel

- We combined them into just the two lines that we really care about.
- we computed the heading direction by simply averaging the far endpoints of both lane lines.





Results:

Challenges We Encountered

There are several limitations:

- Changes in lighting or other environmental conditions.
- Sensitive to the presence of other objects or distractions in the scene.
- While turning the robot the analysis gets confused.

RECOGNIZING TRAFFIC LIGHTS

3-Step Process

1

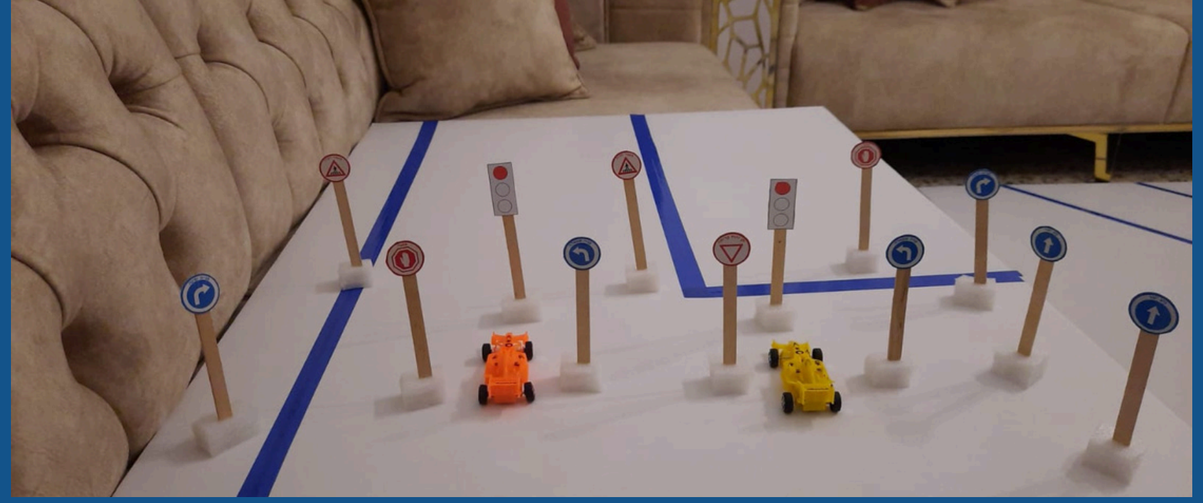
Collecting images of our custom objects and Labeling them

2

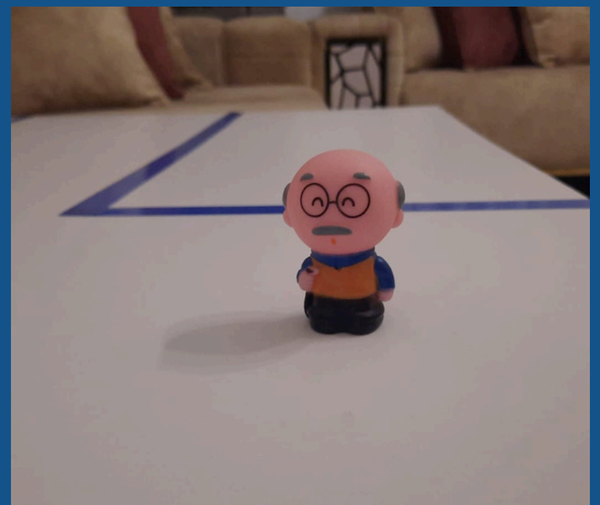
Preprocessing

3

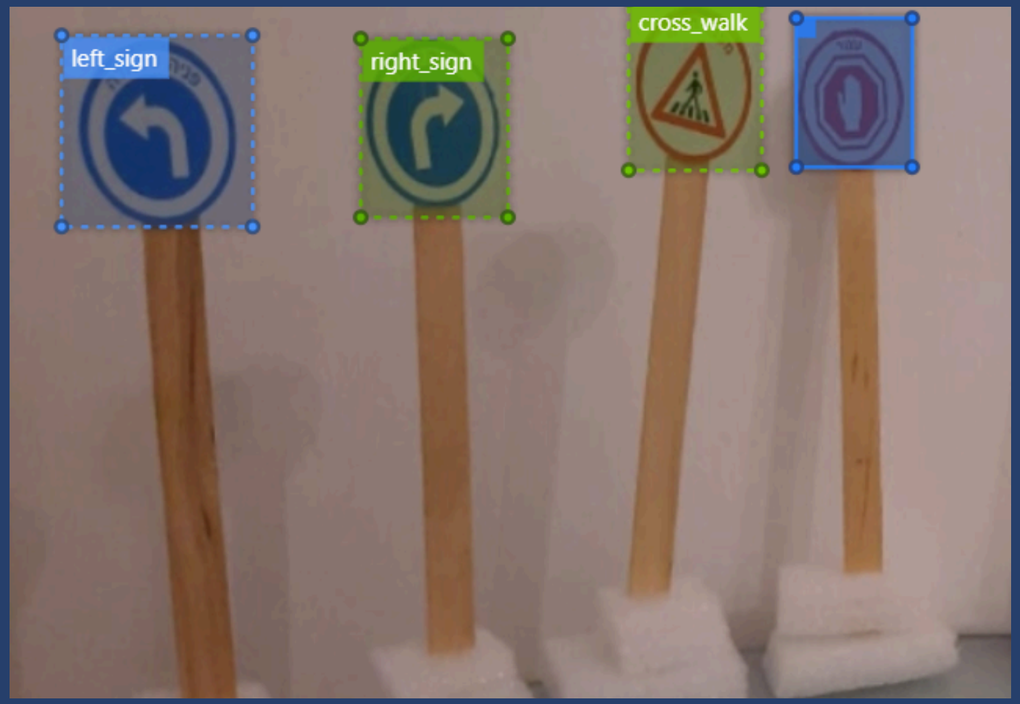
Using transfer learning to train a custom model on our objects

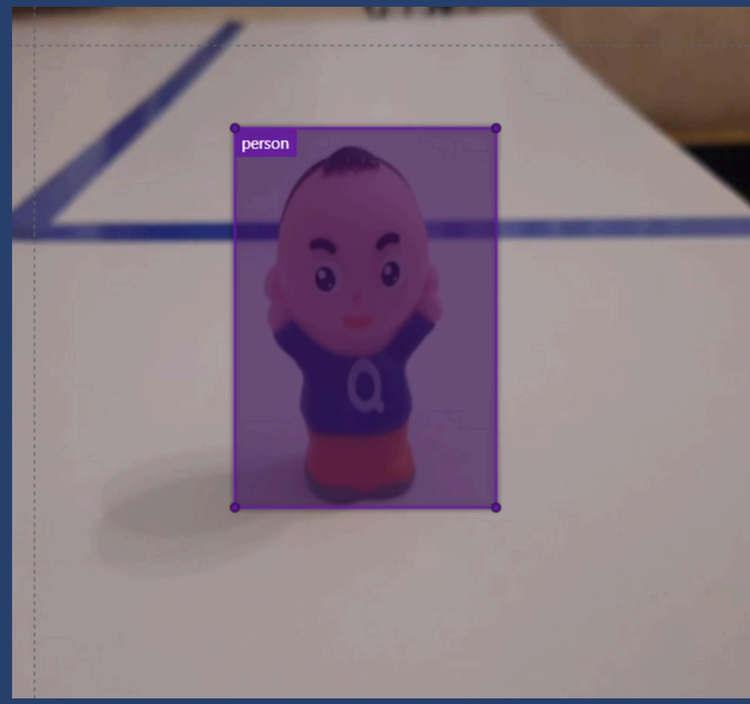


Collecting Data



Labeling the objects with a class





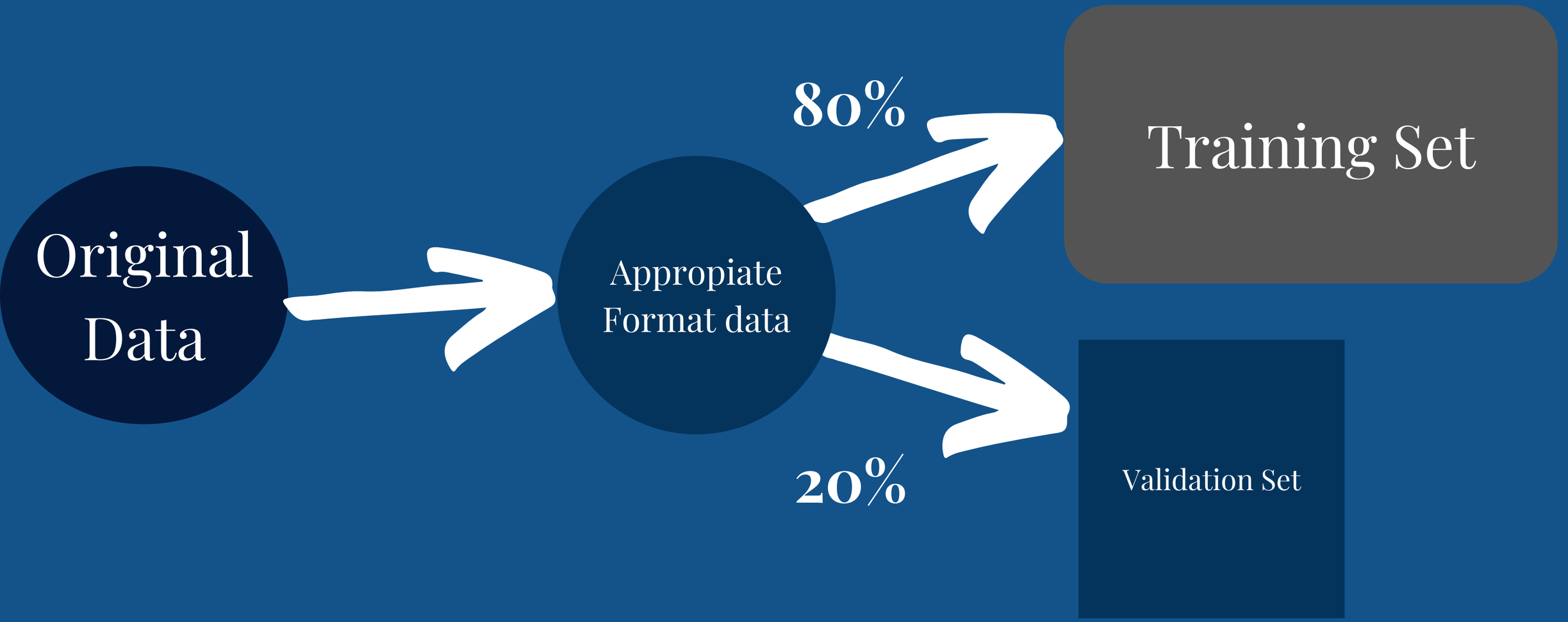
img1.txt

```
0 0.47 0.35833333333333334 0.265 0.1733333333333334
```

class number

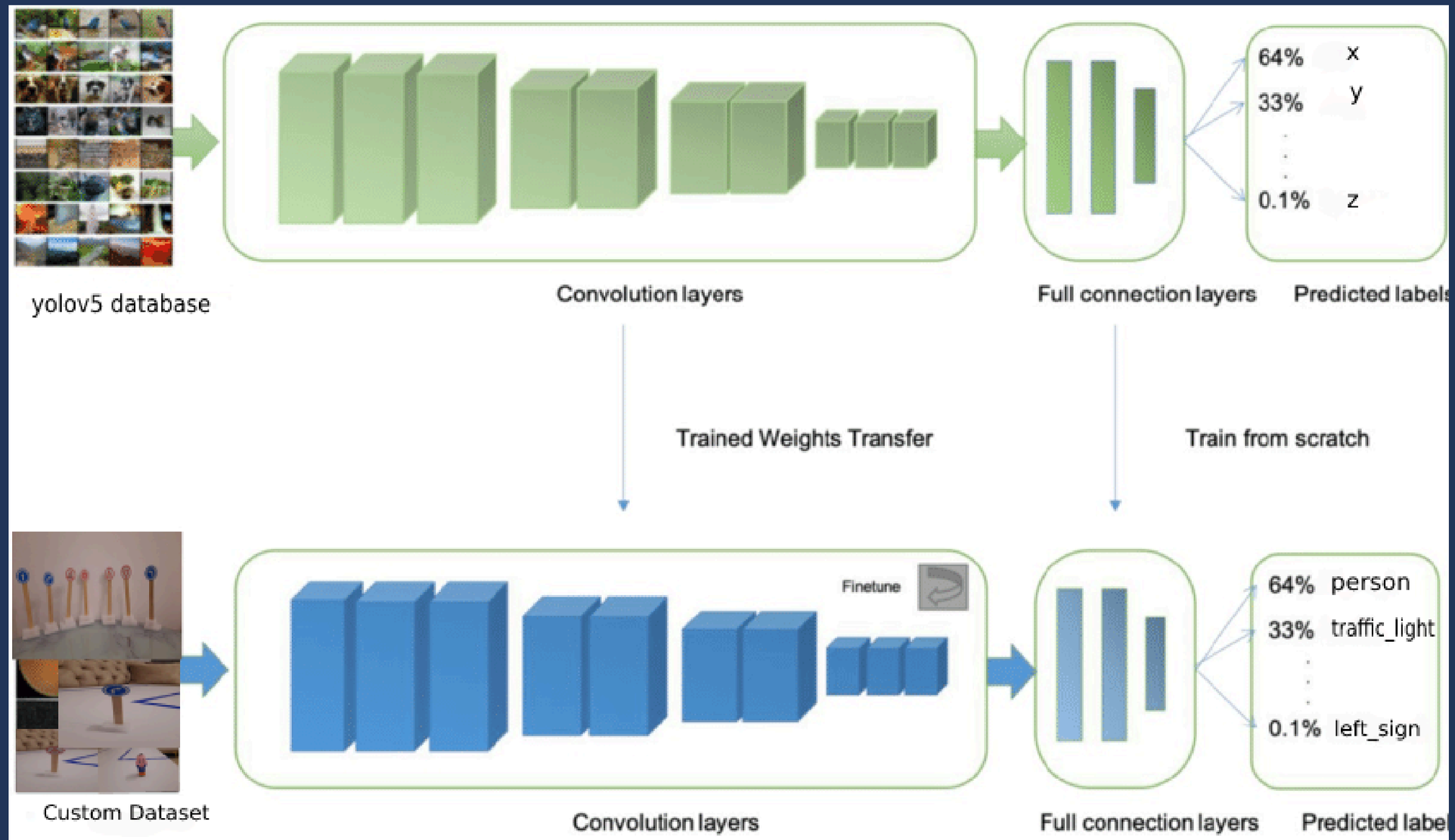
Coordinates

Preprocessing



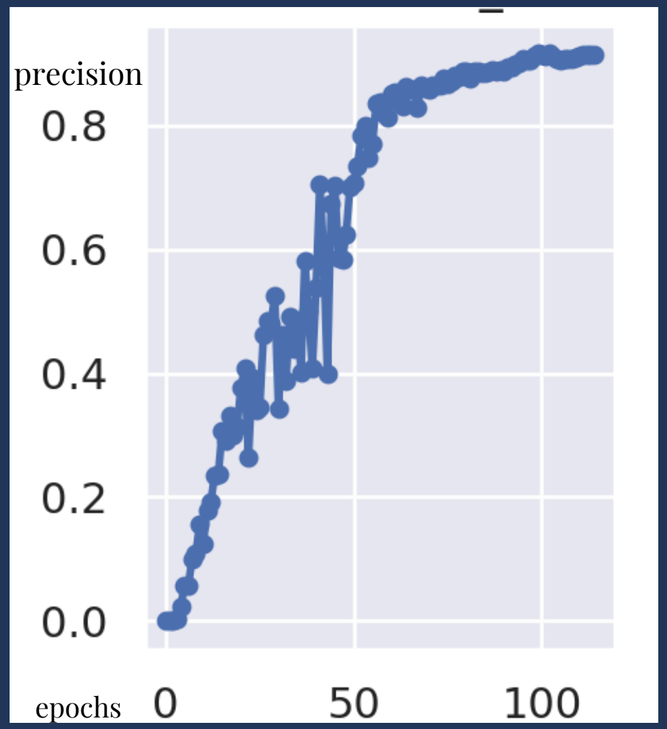
Transfer Learning

- we used YOLOv5 as the base model for our custom model

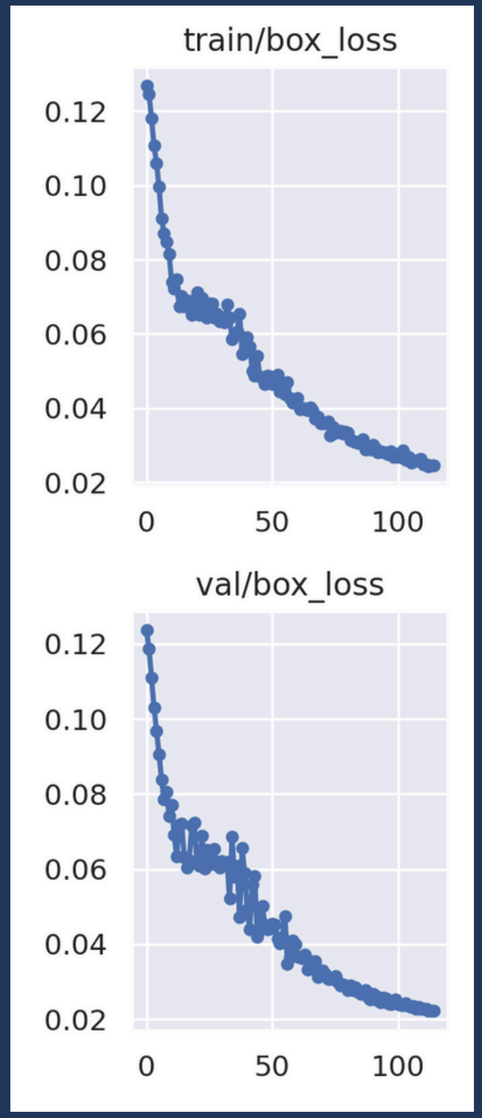


- **Model Results**

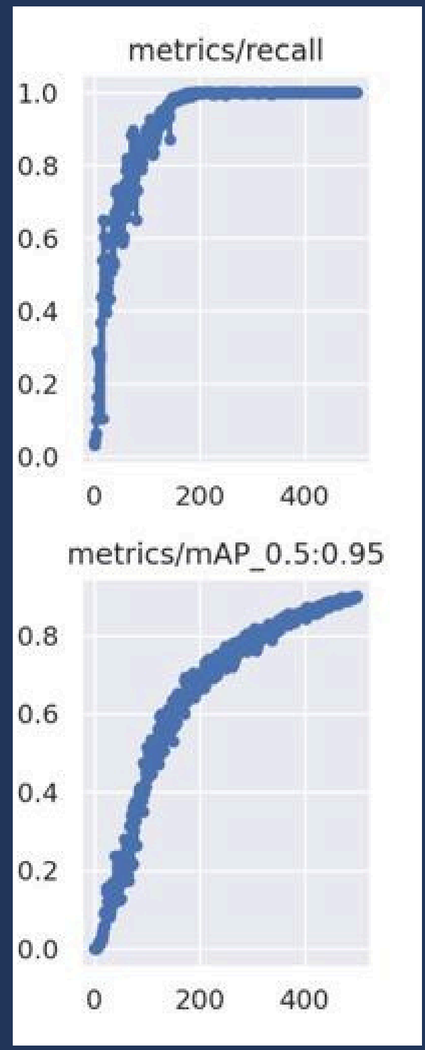
General Accuracy: 94%



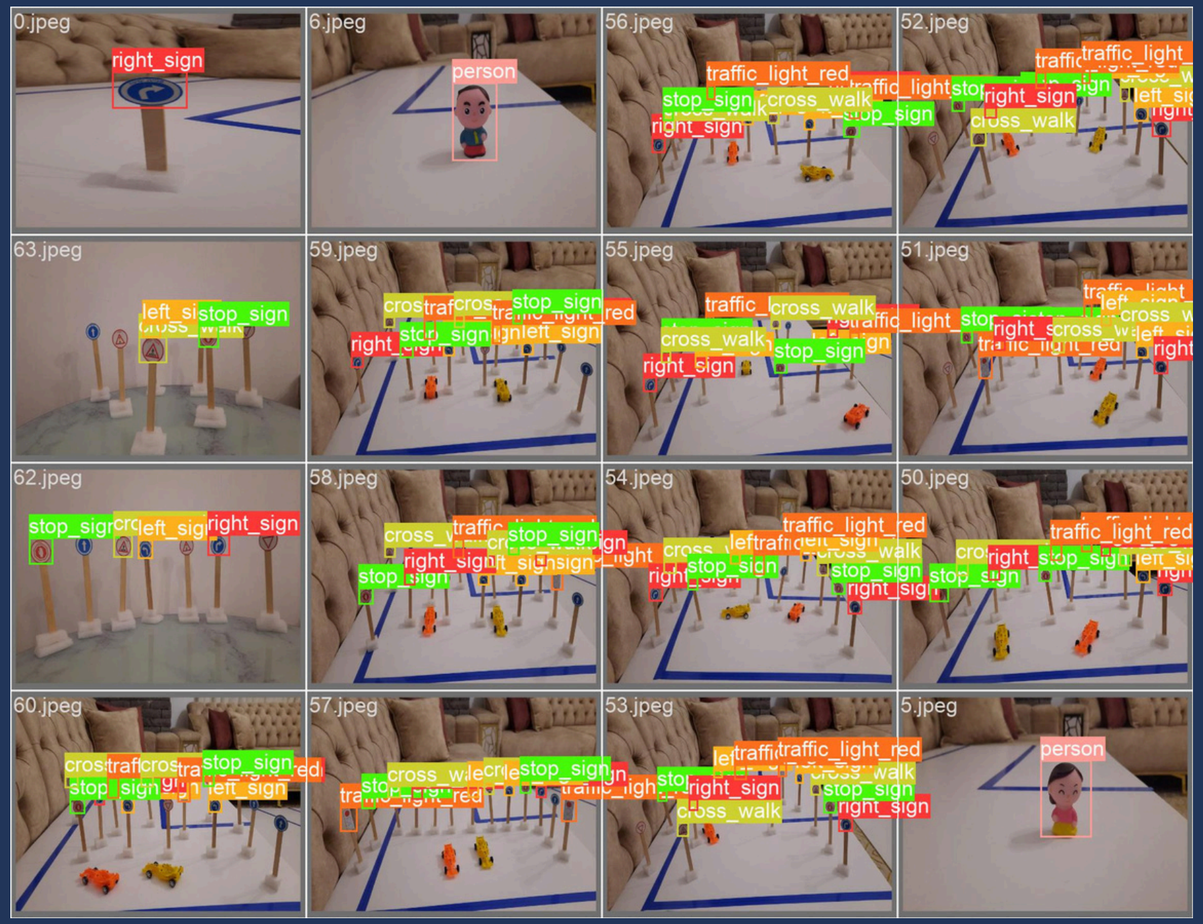
Box loss

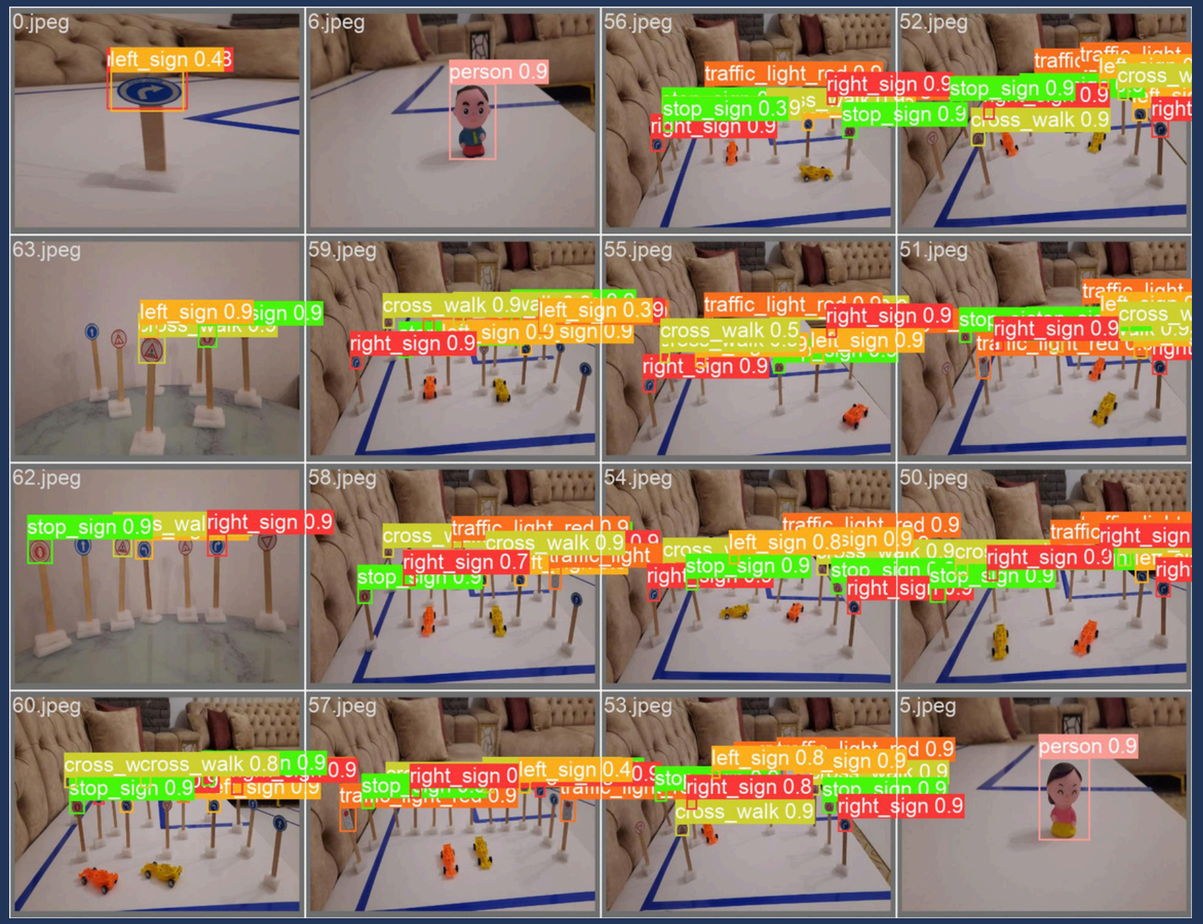


Classification loss



• Model Results



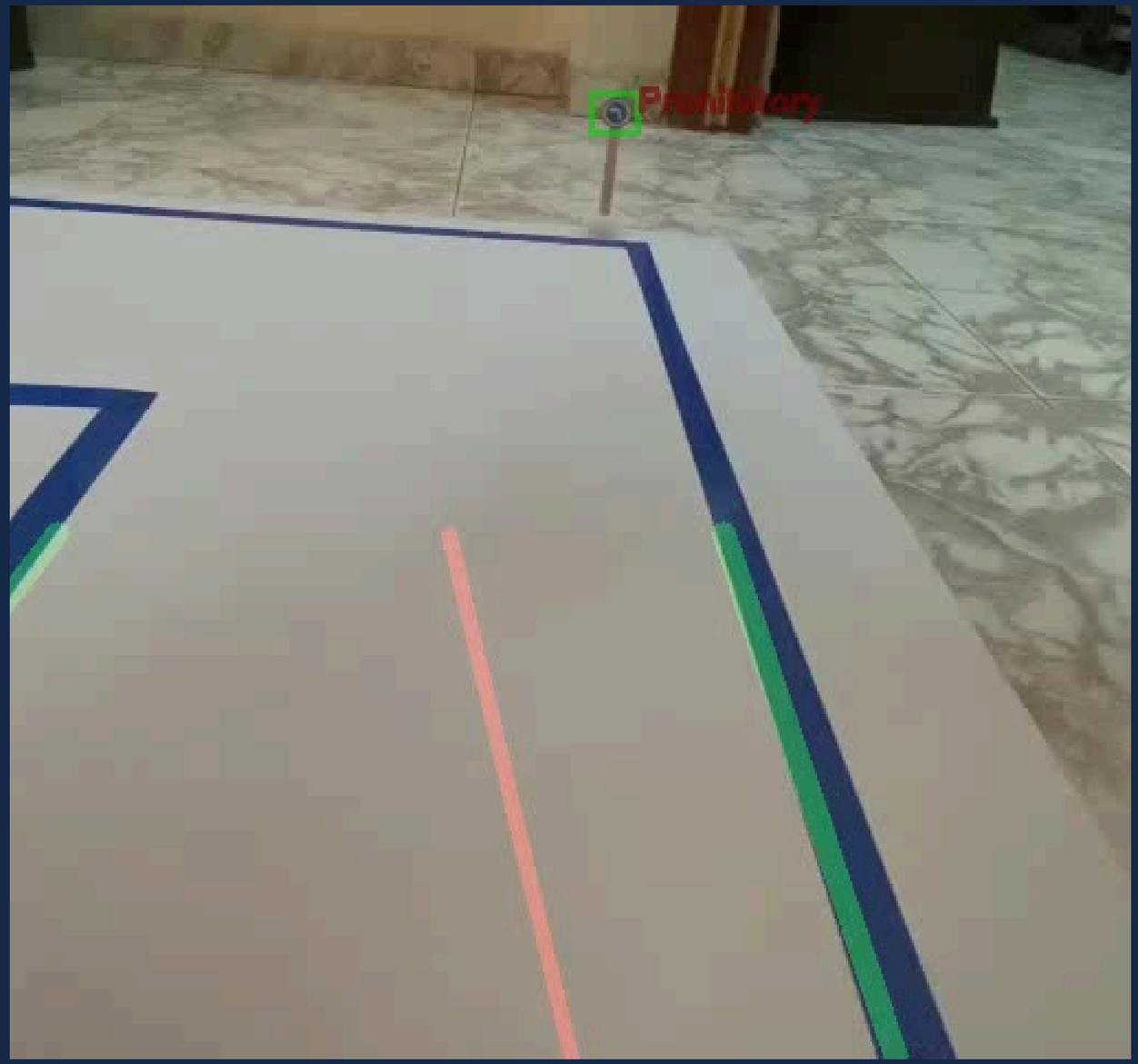


Challenges We Encountered

There are several limitations:

- Changes in lighting or other environmental conditions, which can affect the accuracy of the object detection.
- Sensitive to the presence of other objects or distractions in the scene.
- Occasionally It predicts the wrong label.

Final Result :



Technologies