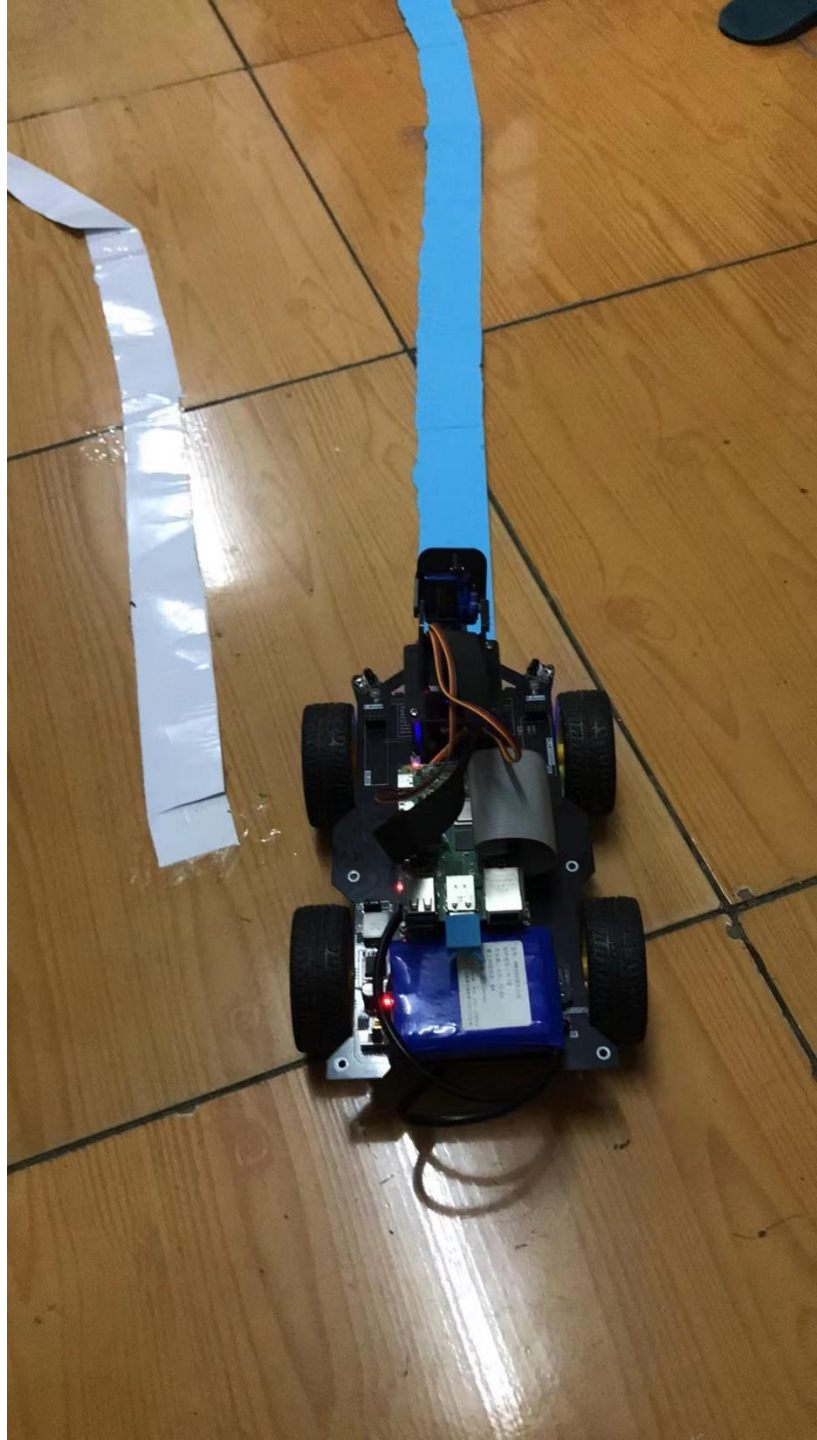
# Blue Color Line Following Car with obstacle detection

using Computer Vision

Team members:

1. Omar 027121126
2. Iskakov 037121119
3. Sabbir 027121125



# Introduction and Importance

Line-following robots have numerous applications in various fields, such as:

1. Manufacturing,
2. Warehousing,
3. Automation.

This experiment focuses on developing a car that can follow a blue line using computer vision and image processing.

The car utilizes a camera to capture images of the ground, and the images are processed to detect the blue line. Based on the position of the line, the car adjusts its speed and direction to stay on the line.

# Objective of the experiment

This experiment explores the implementation of a blue line following car using computer vision and image processing techniques.

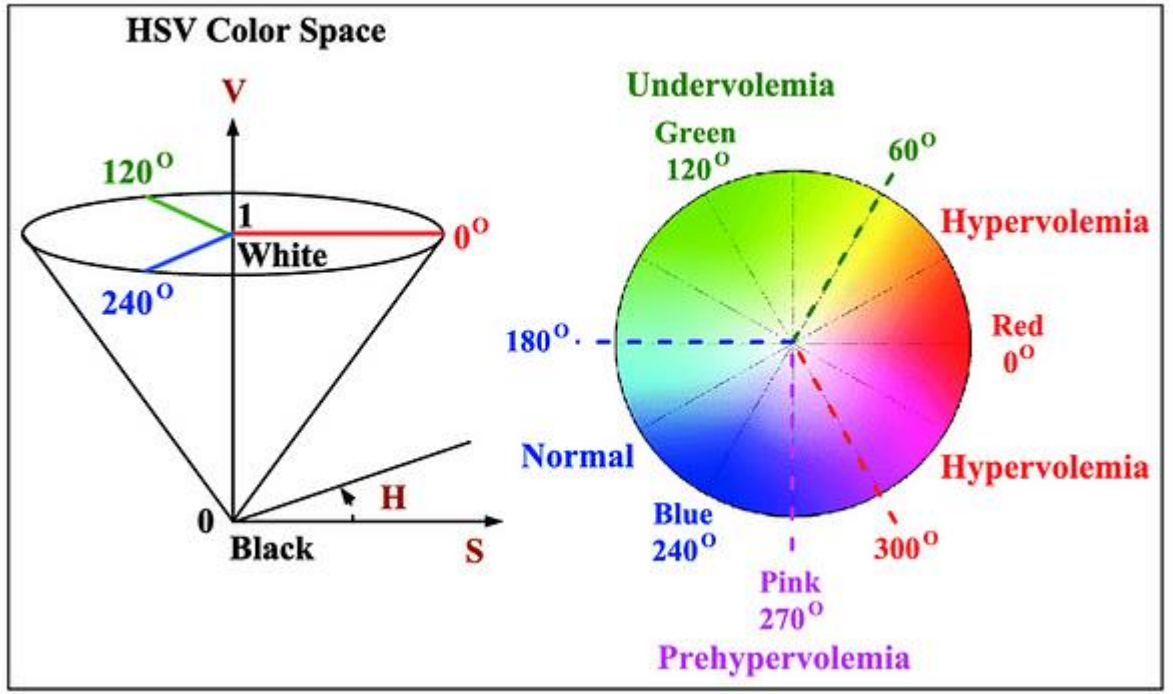
The car is equipped with a camera and a Raspberry Pi to capture and process images in real-time.

The goal is to enable the car to autonomously follow a blue line on the ground by detecting the line and adjusting its movement accordingly.

# Methodology

1. **Image Capture**: The camera captures frames continuously in real-time.

2. **Color Space Conversion**: The captured frame is converted from BGR color space to HSV color space to facilitate color-based segmentation.




H = (G - B) / (max(R, G, B) - min(R, G, B)) * 60
S = (max(R, G, B) - min(R, G, B)) / max(R, G, B)
V = max(R, G, B)

OpenCV library handles the color space conversion internally using optimized algorithms.

# Methodology

**3. Binary Mask:**

We define the range of blue color in HSV using Numpy array. In our case the color is sky blue and the range is as follows:

```
lower_blue = np.array([100, 150, 150])
upper_blue = np.array([140, 255, 255])
```

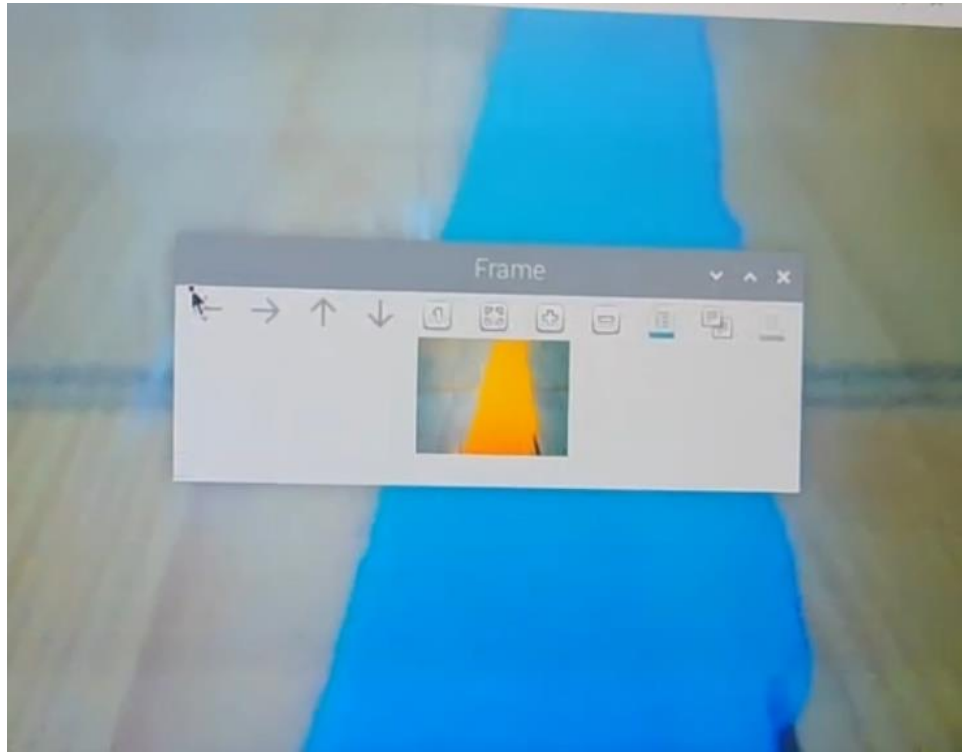
Then we create a binary mask where pixels within the blue color range are set to white (255) and the rest are set to black (0).

# Methodology

**4. Contour Detection**:

Contours are the boundaries of objects or shapes within an image. Contour detection helps identify the path or track that the car needs to follow.

In our experiment, we apply the **cv2.findContours()** function directly on the binary mask. It finds contours in the binary mask, which correspond to the blue line segments in the image.



# Methodology

**Contour:**

0 0 0 0 0
0 1 1 1 0
0 1 1 1 0
0 1 1 1 0
0 0 0 0 0

(0,0) (0,1) (0,2) (0,3) (0,4)
(1,0) (1,1) (1,2) (1,3) (1,4)
(2,0) (2,1) (2,2) (2,3) (2,4)
(3,0) (3,1) (3,2) (3,3) (3,4)
(4,0) (4,1) (4,2) (4,3) (4,4)

overlay the contour pixels onto the coordinate grid

(1,1), (1,2), (1,3)
(2,1), (2,2), (2,3)
(3,1), (3,2), (3,3)

binary image of the contour

image with pixel coordinates

When calculating the moments (M_00, M_10, M_01), we use these pixel coordinates and their corresponding intensity values (1 for contour pixels, 0 for background pixels) to perform the necessary calculations.
M_00 = 1+1+1+1+1+1+1+1+1 = 9

For example, to calculate M_10, we multiply each contour pixel's x-coordinate by its intensity value and sum up the results:
M_10 = (1 * 1) + (2 * 1) + (3 * 1) + (1 * 1) + (2 * 1) + (3 * 1) + (1 * 1) + (2 * 1) + (3 * 1) = 18

Similarly, for M_01, we multiply each contour pixel's y-coordinate by its intensity value and sum up the results:
M_01 = (1 * 1) + (1 * 1) + (1 * 1) + (2 * 1) + (2 * 1) + (2 * 1) + (3 * 1) + (3 * 1) + (3 * 1) = 18

# Methodology

**5. Centroid Calculation**:

The centroid is the geometric center of an object or shape. Centroid represents the center point of the detected track.

The centroid coordinates (cx, cy) are calculated as:
cx = M_10 / M_00
cy = M_01 / M_00

In this example, the centroid coordinates would be:

cx = 18 / 9 = 2,  cy = 18 / 9 = 2

The centroid represents the center of mass or the average position of the contour.

# Methodology

**Center Deviation:**

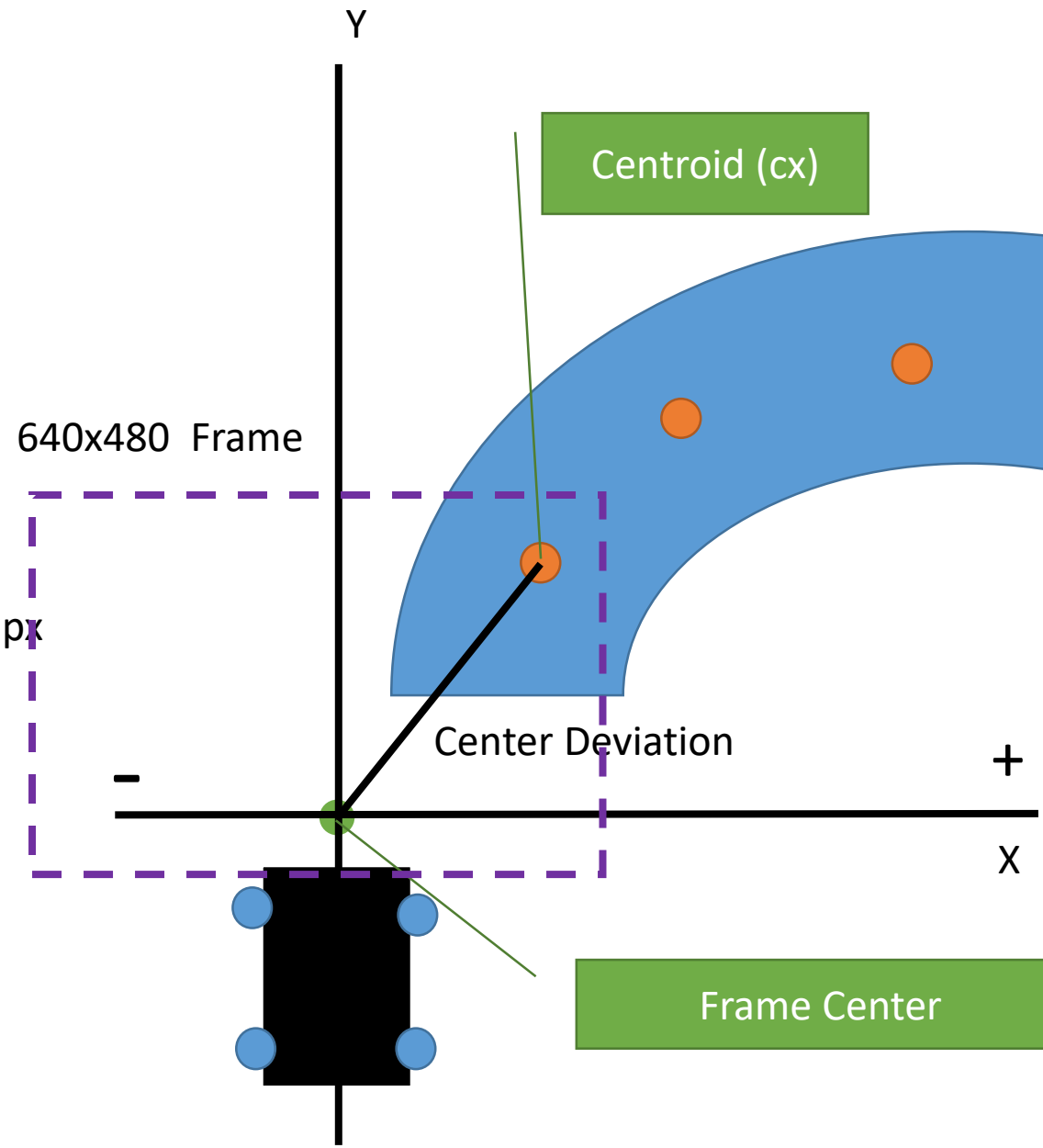
The center deviation represents how far the blue line is from the center of the frame.

center_deviation = cx - frame_center

If the frame is 640x480 px then the frame_center = 320 px

A positive center deviation means the blue line is positioned to the right of the frame center.

Negative means the left and zero indicates that the blue line is perfectly aligned with the frame center.



# Methodology

**Robot Movement Control:**

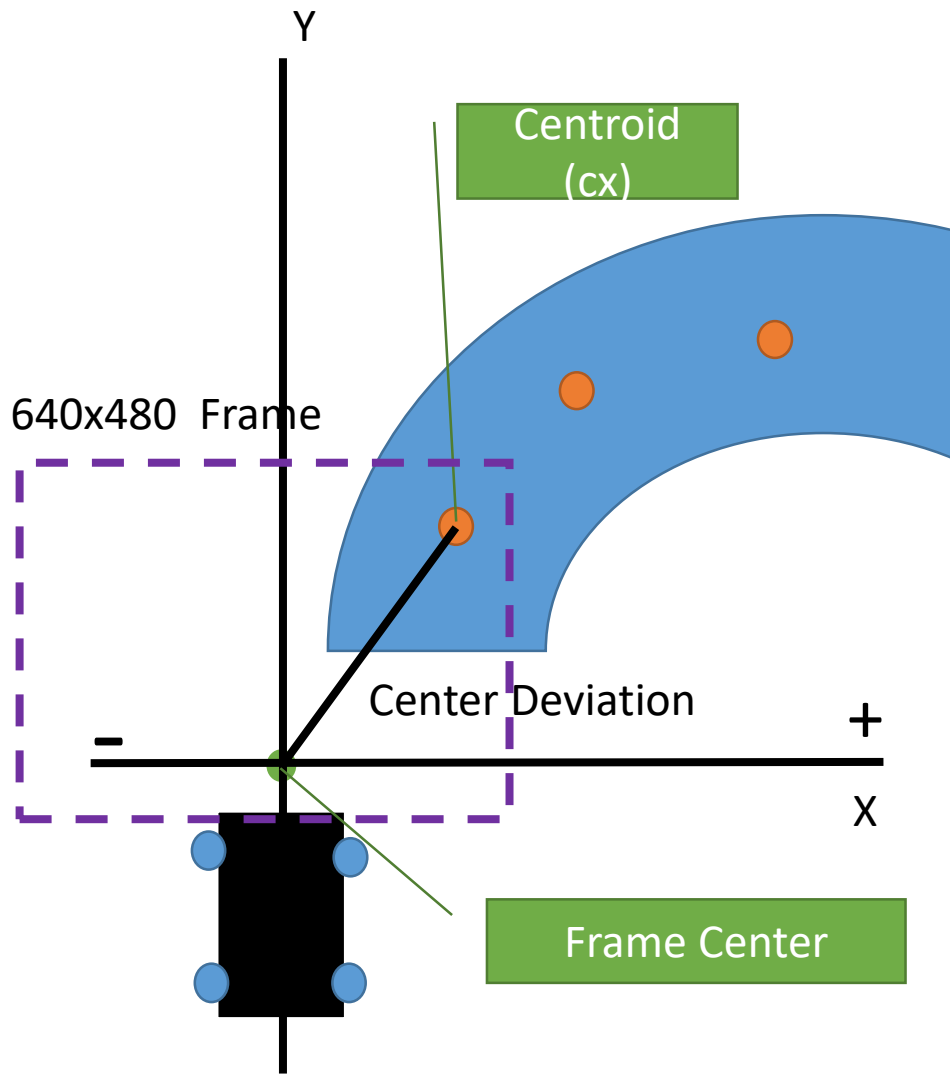
The center deviation plays a crucial role in determining how the robot adjusts its movement to follow the blue line:

1. If the center deviation is **within a certain threshold** (e.g., between -50 and 50), the robot is considered relatively centered on the line, and it continues to move **straight**.

2. If the center deviation is **positive** (greater than the threshold), the robot is instructed to **turn right** by reducing the speed of the **right motor** proportionally to the deviation.

3. If the center deviation is **negative** (less than the negative threshold), the robot is instructed to **turn left** by reducing the speed of the **left motor** proportionally to the deviation.

4. The **larger the absolute value** of the center deviation, the **sharper the turn** the robot will make to align itself with the blue line.



# Methodology

6. **Motor Speed Control**:

The speed of the motors is adjusted based on the center deviation. The speed adjustment is proportional to the deviation, with a maximum reduction equal to the base speed.

The adjusted speeds for turning right and left are calculated as:

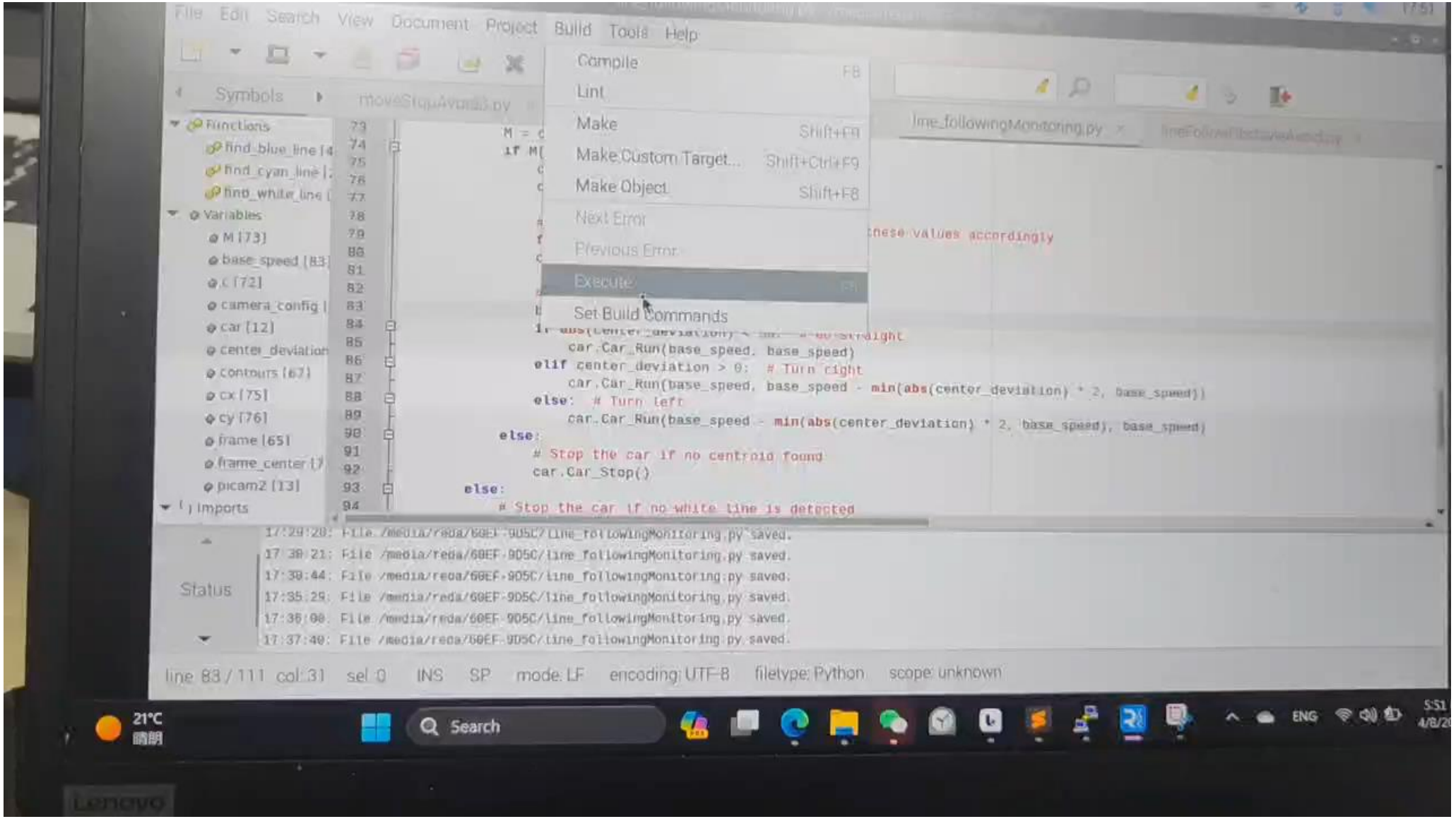
- Right motor speed = base_speed - min(abs(center_deviation) * 2, base_speed)
- Left motor speed = base_speed - min(abs(center_deviation) * 2, base_speed)

This ensures that the speed reduction is proportional to the deviation but does not exceed the base speed.

When the blue line is close to the center, the speed adjustment will be small, allowing the robot to move relatively straight. As the blue line deviates further from the center, the speed adjustment increases, causing the robot to turn more sharply to align itself with the line.

# Results

Here is a live preview of the Raspberry Pi camera:



# Results

## Evaluation of the line following performance:

**Smooth and accurate movement:**
The car successfully follows the line smoothly except for the external lighting. If there is no external reflection of light then the car will follow the line without any error.

**Handling of turns and curves:**
During the test, we set up two curves and the car was able to make the terns in the curve.

Notes: At the beginning, the car follows the light but later it successfully detects the blue line and continues following the blue line. So we need to consider the lighting condition.

# limitations

**1. Lightings:**
The car can’t detect the line when there is low light or external reflection of light.

**2. Camera position**
To get the best result we had to put the camera close to the line.

**3. Dark environment**
The experiment can’t be done in a dark environment. In that case, we can put an external LED on the car or use other sensors to follow the line.

# Conclusion

This experiment showcases the application of computer vision and image processing techniques in the development of a blue line following car. By leveraging the **OpenCV** library and **color-based segmentation**, the car can detect and follow a blue line in real time.

The algorithm proves to be effective in controlling the car's movement based on the position of the line.

Future enhancements to the project could include:

1. The integration of obstacle detection and avoidance.
2. The ability to handle intersections and junctions
3. The implementation of more advanced control algorithms for smoother and more precise line following.

# Questions and Discussion:

Do you have any questions?