M. Huraira Anwer - 25100314 Sadaan Tahir - 25100224

EE 3002 L1 (Junior Design Studio - Robotics)

Spring 2025 - LAB 5

In this lab, we will work with the **RoboMaster EP Core's** camera module. All the tasks should be completed on your regular operating system, as they do not require ROS - only Python 3.6-3.8 and the RoboMaster SDK.

Optional (if your Windows system doesn't already have python 3.6-3.8):
Use the following guide to set up anaconda python virtual environment:
https://medium.com/@abirmunna091/setting-up-a-new-anaconda-environment-and-installing-python-3-8-on-windows-4a1fa748eddf

Some specifications of the embedded camera module are as follows:

- **FOV: 120°**
- Max Still Photo Resolution: 2560×1440
- Max Video Resolution: FHD: 1080p/30fps HD: 720p/30fps
- Max Video Bitrate: 16 Mbps - Photo Format: JPEG
- Video Format: MP4
- Sensor: CMOS 1/4″; Effective pixels: 5 MP spot.
- Operating Temperature Range: -10 to 40 °C (14 to 104 °F)

Task 1: Understanding Robomaster Camera [20 MARKS]

In this task, you will explore the RoboMaster EP Core's camera module and its vision capabilities **by detecting markers and estimating their distance from the robot**.

Steps:

1. Install the RoboMaster SDK if not already, then open a terminal and change the directory to the RoboMaster SDK examples folder: **/examples/05_vision/**.
2. Before running any script, update the connection type to ensure a stable wireless connection: ep_robot.initialize(conn_type="ap").
3. Run the **01_marker.py** script in the **05_vision** folder and observe whether the system successfully detects the provided vision markers. This will help you understand how the **EP Core** processes its camera feed and how we can explore its inbuilt functionality to detect markers. You can test the other scripts in this folder on your own as well.

Note: You may need to add time.sleep(1) after the line **ep_camera.start_video_stream(display=False)** in the **01_marker.py** script to avoid any emptyQueue errors.

4. Now that you can detect markers, **write a Python script** called **marker_detection.py** to estimate the distance of a detected marker from the robot. The script should:
   4.1. **Detect the marker's position in the camera feed**.

4.2. **Calculate the real-world distance from the robot using marker size and focal length**.
4.3. **Display the detected marker along with its coordinates and estimated distance**.

HINTS:

- The distance from the object can be estimated using this formula:
  - **Distance = (real width of marker * focal length) / pixel width of marker**
- The angle of the object relative to the robot can be calculated using:
  - **Angle = (x_pixel - center) * angle per pixel**
- Color masking would be needed for object detection. These links might help:
  https://medium.com/@sasasulakshi/opencv-object-masking-b3143e310d03
  https://pyimagesearch.com/2014/08/04/opencv-python-color-detection/

Task 2: Detecting and Gripping an Object [20 MARKS]

In this task, you will develop a Python script that detects a colored object using image masking, identifies its position using contours, moves the RoboMaster towards it, and attempts to grip it using the robot's gripper.

Implement a **Python script** that detects a colored object, preferably **an object with a uniform color throughout**. Create a mask and define the **min and max RGB ranges** for that color. Apply the mask to the image and set all other pixels to 0. Now detect contours and pick the largest contour and get its x, y coordinates, width, and height in pixels. Using these x, y coordinates of the object from the robot, move towards that object and grip it using the robot gripper.

Since the estimated x, y coordinates and distance to the object may not be perfectly accurate due to noise or detection errors, the robot should move towards the marker gradually instead of trying to reach it in one step. For example, if the initial estimated distance is 0.8m, the robot moves a smaller step, such as 0.2m, then re-evaluates the distance. If the new estimate is 0.7m, it moves another smaller step, like 0.15m, and continues this process until it reaches a minimum distance threshold (e.g., 0.1m). This incremental approach will help in correcting errors, prevent overshooting, and ensure smoother and more precise navigation towards the object.

Refer to the script **/examples/11_gripper/01_open_close.py** in order to understand how to use the gripper manually. Also, since some objects might not be easy to grip, as long as it shows that it is opening the gripper once it has approached the object, you will receive marks for the part. However, consider it a personal achievement if you manage to actually grip the object by the gripper and bring it back to the robot's initial starting position.

Steps:

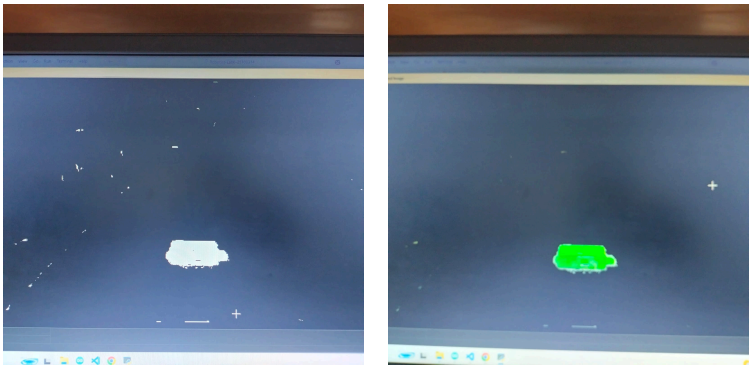
1. Capture an image from the robot's camera for processing
2. Convert the image from **BGR to HSV** color space.
3. Define the **lower and upper HSV bounds** for the **target** object's **color**.
4. Create a **binary mask** where the object appears white and the background is black.
5. Extract contours from the mask and **identify the largest one**.

6. Get the bounding box coordinates **(x, y), width, and height** of the largest contour.
7. Estimate the object's **real-world distance using the bounding box width**.
8. Compute the angle to the object based on the **camera's FOV**.
9. Move towards the object in **small increments**. Continue moving until the robot reaches a predefined **minimum distance threshold**.
10. Rotate the robot to align it with the object if necessary.
11. Open the gripper once within the gripping range. Grip the object and go back to the **initial starting position** and release the object from the gripper.

HINTS:

- Here are some resources to understand **contours and object detection in Python**:
  https://copyprogramming.com/howto/bounding-box-on-objects-based-on-color-python
  https://learnopencv.com/contour-detection-using-opencv-python-c/

Here are some pictures that depict what the mask should look like (**left**) and what the image should look like after the mask has been applied to it (**right**):



Also, the largest contour detection should give results as shown below. The **left image** shows the **largest contour detected for the green mask** and the **right image** shows the **masked largest contour**.

Task 3: Lane Following [10 MARKS] + 10 Bonus Task

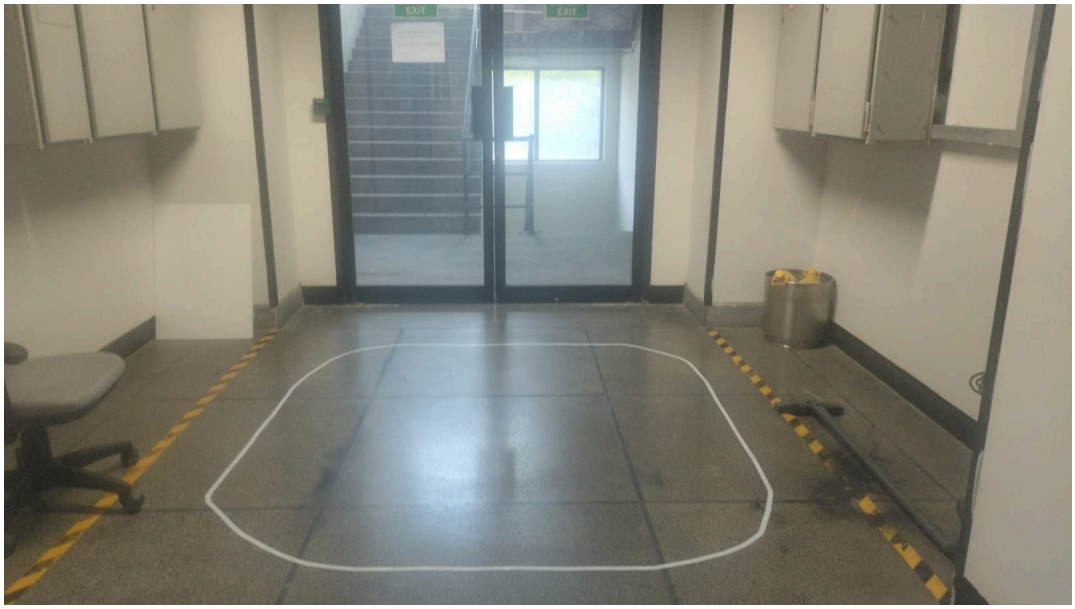
- Implement Lane following robots. Program a controller for your robot that detects lanes using images, and find the angle of the lane, and then feed that angle to your robot using angular velocity, to follow the lanes.
- **Steps Breakdown:**
    - Read the image from the robomaster camera.
    - Crop the Region of interest, (You can constantly see robot grippers in the image, that is unwanted information and can act as noise, so crop out the lower region of the image.
    - Color masking, define the color ranges for the line, and remove all other colors from the image.
    - Edges detection, detect edge in the masked imaged using edge detection algorithm such as canny edge detection.
    - Find the line in an edged image using line Hough transform.
    - Find the angle of the line and feed to your controller.

Task Requirement :

Show line following on a simple line curve with at least one turn in the curve.

Bonus Task :

There is a Circular track at the end of the corridor near the lab entrance, completing one lap for getting full bonus marks.



Note:
This is an Open problem; you can do this task using any method (use a machine learning model) or completely mathematical approach. It's up to you.

Useful resources.

- https://const-toporov.medium.com/line-following-robot-with-opencv-andcontour-based-approach-417b90f2c298
- https://www.hackster.io/Aasai/lane-following-robot-using-opencv-da3d45

You can also take a look at robomaster sdk for line detection in **05_vision>02_line** Use these resources carefully, they may have some errors, and you need to fine tune them for your use case.

Submission Requirements:

1. Include a **zip file** containing the following items:
   1.1. **All the python scripts implemented**.
   1.2. **Videos** showcasing working python codes **for Task 2**.and **Task 3.**
2. Submit a **LAB report** in PDF format (please do not submit word files). This report should provide explanations for all tasks performed along with screenshots, whenever suitable.