M. Huraira Anwer - 25100314 Sadaan Tahir - 25100224

EE 3002 L1 (Junior Design Studio - Robotics)

Spring 2025 - LAB 4

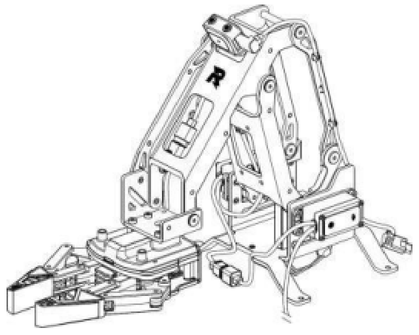
In this lab, we will use the **RoboMaster EP Core** as our primary learning platform. The complete specifications for this DJI robot can be found here:
https://www.dji.com/global/support/product/robomaster-ep-core

Hardware

The following is an overview of the electronic components that comprise the EP Core:

EP Core

- Weight = approx. 3.3 kg
- Dimensions = 320×240×270 mm (length × width × height)
- Chassis Speed Range = 0-3.5 m/s (forward) , 0-2.5 m/s (backward) , 0-2.8 m/s (sideways)
- Max Chassis Rotational Speed = 600°/s



Robotic Arm and Gripper

- Movement Range: 22 cm (horizontal), 15 cm (vertical)
- Number of Axes: 2
- Gripper Range: approx. 10 cm

Intelligent Controller

- Latency: Connection via Wi-Fi: 80-100ms Connection via Router: 100-120ms
- Live View Quality: 720p/30fps
- Max Live View Bitrate: 6 Mbps
- Operating Mode:Frequency: 2.4 Connection GHz, 5.1 via Wi GHz, 5.8-Fi and Router GHz

Infrared Sensor

- Range: 0.1-10 m
- Field of View = 20°
- Accuracy = 5%

Power Module Adapter

- Communication Port: Controller Area Network (CAN) bus
- Output: USB Type-A power port: 5V 2APower port with pin header: 5V 4ATX30 power port: 12V 5A
- Input: TX30 power port: 12 V

Camera

- FOV: 120°
- Max Still Photo Resolution: 2560×1440
- Max Video Resolution: FHD: 1080/30fps HD: 720/30fps
- 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)

Intelligent Battery

- Capacity: 2400 mAh
- Nominal Charging Voltage: 10.8 V
- Maximum Charging Voltage: 12.6 V
- Battery Type: LiPo 3S
- Energy: 25.92 Wh
- Weight: 169 g
- Operating Temperature Range: -10 to 40 °C (14 to 104 °F)
- Charging Temperature Range: 5 to 40 °C (41 to 104 °F)
- Maximum Charging Power: 29 W
- Battery Life in Use: 35 minutes (measured at a constant speed of 2 m/s on a flat surface)
- Battery Life on Standby: Approx. 100 minutes

Charger

- Input: 100-240 V, 50-60Hz, 1A
- Output: Port: 12.6 V=0.8A or 12.6 V=2.2A
- Voltage: 12.6 V
- Rated Voltage: 28 W

Software

Next, we use the **RoboMaster SDK** to interface with the EP Core's sensors and motors, enabling control and data retrieval. The **RoboMaster SDK** is a collection of development tools designed for DJI RoboMaster series products, including the RoboMaster EP, EP Core, Tello EDU, and Tello Talent. It enables users to control robot movements from a PC and access data from the robot's sensors. For more info, visit:
https://robomaster-dev.readthedocs.io/en/latest/index.html

Additionally, the **RoboMaster App** can be installed using the links below (this is purely for fun):

- Android application (SDK): https://www.dji.com/global/downloads/djiapp/robomaster
- Desktop application: https://www.dji.com/global/downloads/softwares/robomaster-win

Finally, connect your RoboMaster EP Core to your PC or phone via Wi-Fi. The Wi-Fi network name (SSID) and password can be found on the robot's router. **On your PC, open Wi-Fi settings and connect to the network using the credentials**. Once connected, launch the RoboMaster app and ensure the connection is stable. Switch to Solo mode within the app and use the keyboard controls to maneuver the robot, verifying that it responds correctly to movement commands.

M. Huraira Anwer - 25100314 Sadaan Tahir - 25100224

Task 1: RoboMaster SDK Setup [20 MARKS]

Steps:

1. **Ensure Python 3.7 is installed** on your laptops or the lab computers because the RoboMaster SDK was primarily developed and tested with this particular version. Other python versions may not work due to compatibility issues. Follow the instructions in the link below for installing python:
   https://robomaster-dev.readthedocs.io/en/latest/code_env_setup.html

2. **Clone and extract** zip file of this GitHub repo:
   https://github.com/dji-sdk/RoboMaster-SDK

You can find detailed installation steps of the RoboMaster SDK for different operating systems in the following link (inclusive of MacOS):
https://robomaster-dev.readthedocs.io/en/latest/python_sdk/installs.html

Task 2: Understanding Robomaster [20 MARKS]

Steps:

1. Once the RoboMaster SDK has been installed, open a terminal and change the directory to the RoboMaster SDK examples folder, **/examples/02_chassis**.

2. Before running any script, update the connection type to ensure a proper wireless connection. Modify this line in the **01_move.py** and **03_speed.py** scripts: ep_robot.initialize(conn_type="sta"). Change the connection type from **'sta'** to **'ap'** always.

3. Run both movement scripts, **01_move.py** and **03_speed.py**. Observe how the robot responds and note the differences between movement types.

4. Run the **04_sub_attitude.py** and **05_sub_position.py** and **06_sub_imu.py** scripts in the **/examples/02_chassis** folder to understand the nature of the callback functions and how RoboMaster uses them to publish position and angular (odometry) as well as imu data at a certain rate. Don't forget to change the connection type here as well. You will need this for Task 4.

5. Implement a **keyboard-based teleoperation Python script**, called **keyboard.py**, by first installing the required keyboard library using pip install keyboard. Utilize functions from either **01_move.py** or **03_speed.py** to control the robot's movement. The script should continuously listen for keyboard inputs and execute movement commands based on **valid key presses (W/A/S/D)** to move the robot **0.5m or 1m** in the **N/S/W/E** directions while ignoring any invalid inputs. Additionally, **include an option to exit the program by pressing Q**. Once implemented, run the script and verify that the robot responds correctly to keyboard inputs.

Task 3: Go-to-Goal (using PID) [15 MARKS]

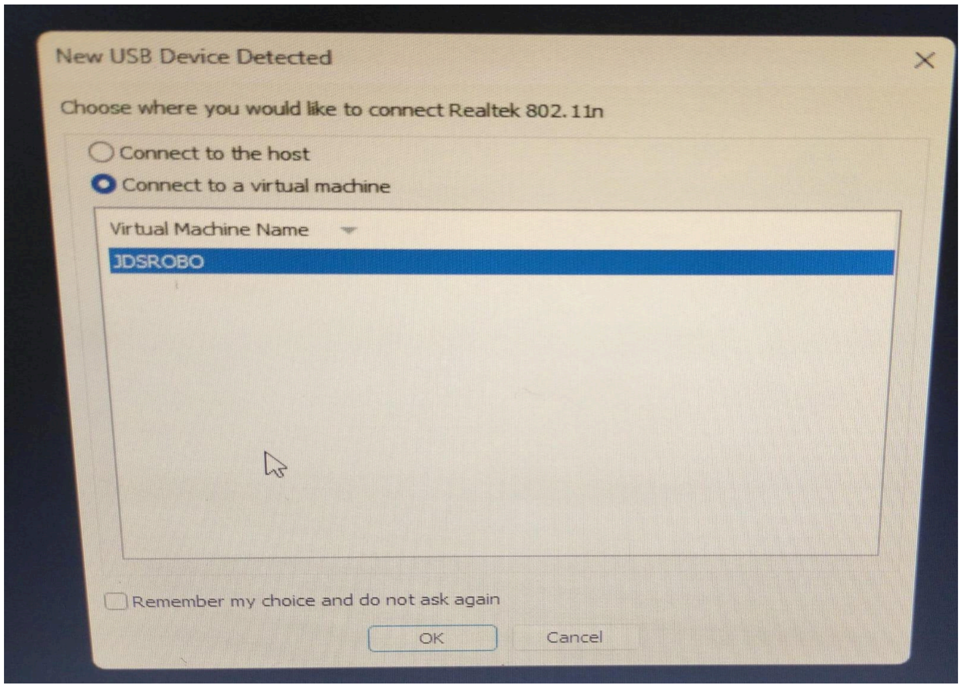
Implement a **Python script**, called **go2goal_PID.py**, that writes a PID based controller that drives the robot to a user-defined goal point, with no obstacles in its path. Consider the following pseudocode for the PID logic, you may or may not need to change/tweak this a little.

```
$\text{current\_location} = [0, 0]$
$\text{goal\_location} = [gx, gy]$
$\text{current\_angle} = \theta$
$dx = \text{distance\_tolerance}$
$\text{distance} = \text{euclidean\_distance}(\text{current\_location} - \text{goal\_location})$
$v_0 = \text{linear\_velocity}$

$\text{PID}(error) = kp \times error + kd \times \frac{d}{dt}(error) + Ki \times \int error$

$\text{while } (\text{distance} > \text{distance\_tolerance})$
$\text{angle\_error} = \arctan\left(\frac{gy - \text{current\_location\_y}}{gx - \text{current\_location\_x}}\right) - \theta$
$\text{angular\_velocity} = \text{PID}(angle\_error)$
$\text{robot}(\text{linear\_velocity}, \text{angular\_velocity})$
```

Task 4: Odometry and IMU Sensor Data [15 MARKS]

For this part, you should be able to connect to wireless wifi. If not, that means the driver is missing, so install it following these steps:

- Plug in Wi-Fi dongle and connect it to your VM - you will see the setting below pop up, choose 'connect to VM' and click on **JDS ROBO.**



New USB Device Detected

Choose where you would like to connect Realtek 802.11n

Connect to the host [✓ Connect to a virtual machine]

Virtual Machine Name
JDSROBO

Open a terminal in the VM and run the following commands

- sudo apt update
- sudo apt upgrade
- sudo apt install net-tools
- sudo add-apt-repository ppa:kelebek333/kablosuz
- sudo apt-get update
- sudo apt-get install rtl8188fu-dkms
- echo "options rtl8188fu rtw_ips_mode=0" | sudo tee /etc/modprobe.d/rtl8188fu.conf
- sudo modprobe -rv rtl8188fu && sudo modprobe -v rtl8188fu
- Remove and reconnect the dongle (if it doesn't still show wireless wifi)

Once it's connected to the robomaster wifi, choose connection type 'ap' and write a script to publish odometry and imu data in ros. As you move the robot, its odometry data should be updated accordingly. You'll need the following imports …

import robomaster
from robomaster import robot
from nav_msgs.msg import Odometry
from sensor_msgs.msg import Imu
from geometry_msgs.msg import Quaternion

Once, you get the odometry and imu data from the robomaster callback functions, you need to publish them as a rostopic .In order to understand Odometry and Imu messages and what they entail respectively, visit these links:
https://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/Imu.html
https://docs.ros.org/en/noetic/api/nav_msgs/html/msg/Odometry.html

These are the values you will be using to set frame ids of the nodes, this is essential for ros to understand which nodes / what data it is dealing with …
odom.header.frame_id = 'odom'
odom.child_frame_id = 'base_link'
imu.header.frame_id = 'imu'

For now, you may set all the **covariance arrays** in IMU message to [0.1,0,0,0,0.1,0,0,0,0.1]. Then, set the **pose.covariance matrix** in Odometry message to a diagnol matrix (with 0.1s).

Use the following function to convert the Euler angles to a quaternion, whose output (qx,qy,qz,qw) you will be assigning to orientation.x , orientation.y, orientation.z and orientation.w respectively.

```
def get_quaternion_from_euler(roll, pitch, yaw):
  """
  Convert an Euler angle to a quaternion.

  Input
    :param roll: The roll (rotation around x-axis) angle in radians.
    :param pitch: The pitch (rotation around y-axis) angle in radians.
    :param yaw: The yaw (rotation around z-axis) angle in radians.

  Output
```

```
    :return qx, qy, qz, qw: The orientation in quaternion [x,y,z,w] format
  """
  qx = np.sin(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) - np.cos(roll/2) *
np.sin(pitch/2) * np.sin(yaw/2)
  qy = np.cos(roll/2) * np.sin(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) *
np.cos(pitch/2) * np.sin(yaw/2)
  qz = np.cos(roll/2) * np.cos(pitch/2) * np.sin(yaw/2) - np.sin(roll/2) *
np.sin(pitch/2) * np.cos(yaw/2)
  qw = np.cos(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) *
np.sin(pitch/2) * np.sin(yaw/2)
  return [qx, qy, qz, qw]
```

Rest of the values in both the messages are pretty straightforward, however you will need to calculate **twist.twist.linear.x, twist.twist.linear.y, twist.twist.angular.z** in the odometry message using the change in **x, y, theta** over time.

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 3**.
   1.3. Screenshot for messages printed in terminal for **Task 4.**

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.