Coding exercise 1 for CS 498 GC

- ☐ This homework consists of a written and coding part. Both must be submitted individually at Gradescope.
- ☐ The coding part must be developed in python with ROS2. When submitting this section, only upload the python files *coding_ex1_part1.py, coding_ex1_part2.py, utils.py, results_part1.txt* and *results_part2.txt* (see attached templates).
- ☐ The written part must be submitted in PDF format. When submitting this file, Gradescope will ask you to select the pages where each question is. Please do it precisely.

Learning outcomes

- ☐ Process data collected from a mobile robot using Robot Operating System (ROS)
- ☐ Understand the concepts of ROS's nodes, topics, and messages.
- ☐ Understand the challenges of dead reckoning for position using velocity data
- ☐ Understand how to plot the data using RVIZ2 and ROS2.

Description

In this coding laboratory, we will process data collected from the Terrasentia ground robot (Figure 1). This is a four-wheeled differential drive robot, equipped with sensors like cameras, LiDAR, IMU, wheel encoders, and GPS.

Figure 1. Terrasentia robot

Question 1. The teaching team collected data from Terrasentia robot over a field. The data collected includes GPS data, encoder velocity, and inertial sensor data. Play the rosbag file and make a list of the topics in your report, mentioning their publishing rate and the type of message used by each topic.

Question 2. Create a ROS node to read the messages from the rosbag file and determine the robot's position along the trajectory. Publish an odometry ROS message which includes the position, the linear and angular velocities, and the heading (in quaternions representation) of the robot.
For this part, we provide two python templates called *coding_ex1_part1.py* and *utils.py*. You must complete the following tasks:

- Create a ROS package and build it with the given python templates (see Tutorial_1 in the appendix section of this document).
- Complete the function *quaternion_from_euler* in *utils.py*.

- Complete the constructor (__init__(self):) of the *odometry_node* class, creating the subscribers to each of the topics in the rosbag file (see example in the template). Write the corresponding callback functions for each subscriber.
- Complete the function *timer_callback_odom*, which is responsible for publishing the odometry message.
- Complete the function broadcast_tf, which is responsible for publishing the transformation between two coordinate frames.

Finally, run your node and play the rosbag file. Visualize the trajectory made by robot using RVIZ2. Add a screenshot of this trajectory to your report. After running your node (for at least 30 s), a .txt file will be saved in your ROS workspace with the name *results_part1.txt*. Submit this file to Gradescope, along with the files *coding_ex1_part1.py*, and *utils.py*.

Question 3. Create a copy of your main template and name it *coding_ex1_part2.py*. Replace the name of the results file in line *self.file_object_results = open("results_part1.txt", "w+")* by *results_part2.txt*.

In this part, publish the same odometry message but instead of computing the position based on the wheel odometry, use the GPS measurements. You can use the function *lonlat2xyz(lat, lon, lat0, lon0)* (in *utils.py*) to compute the displacement in east-west direction (x) and north-south direction (y) with respect to a fixed point with coordinates (lat0, lon0). Choose wisely the initial coordinates, so that x and y start at zero value. Uses RVIZ2 to visualize the final trajectory and add it to your report.
Submit the files *coding_ex1_part2.py*, *utils.py* and *results_part2.txt* through Gradescope.

Question 4. Compare the results from questions 2 and 3. Explain the reasons for any similarities or differences.

Evaluation
coding. 120 points
Report. 30 points

Appendix
Tutorial 1. Building a ROS package
This tutorial is intended for ubuntu 22.04 and ROS2 Humble. For further information go to the official tutorials, where you can find other alternatives.
ROS Humble. https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Colcon-Tutorial.html
ROS Foxi. https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Colcon-Tutorial.html

- Create a ROS workspace if you have not created one yet

```
>> source /opt/ros/humble/setup.bash
>> mkdir -p ros2_ws/src
```

- Create a package in your ROS workspace named *mobile_robotics*.

```
>> cd ros2_ws/src
>> ros2 pkg create --build-type ament_python mobile_robotics
```

- Put the python templates in the directory …ros2_ws/src/mobile_robotics/mobile_robotics/
- Edit the *package.xml* file adding the dependencies rclpy, std_msgs, geometry_msgs, nav_msgs, tf2_ros.

```
<license>TODO: License declaration</license>
  <exec_depend>rclpy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>geometry_msgs</exec_depend>
  <exec_depend>nav_msgs</exec_depend>
  <exec_depend>tf2_ros</exec_depend>
```

- Add a new console script in the file *setup.py* of your package as follows

```
'console_scripts': [
        'part1 = mobile_robotics.coding_ex1_part1:main',
    ],
```

- Build your package

```
>> sudo apt install python3-colcon-common-extensions //if you have not installed it yet
>> cd ros2_ws
>> colcon build --packages-select mobile_robotics
```

- Source your ROS workspace and local workspace in a new terminal and run your node

```
>> cd ros2_ws
>> source /opt/ros/humble/setup.bash
>> . install/local_setup.bash
>> ros2 run mobile_robotics part1
```

- You can play the rosbag file in other terminal

```
>> ros2 bag play solar_house
```