Report: Test task for ROS developer

<u>Introduction:</u> This is a report on a ROS task I implemented in the following environment:

ROS version: ROS 2 Foxy, Robot: Neobotix MPO-500, Simulation: Gazebo.

This report explains the decisions I made based on the options I could use from the environment setup mentioned above. Besides that, I also mentioned several improvement scopes and some technical difficulties I faced which might have hindered the optimization of the program and performance. The tasks are:

1. Measure the total distance the robot has travelled in meters. This information should be logged every time the robot moves by teleoperation or autonomous driving. Output this information in a single json file and keep it updated. Note that the robot is an omnidirectional robot, which can also move holonomically.
2. Log the paths which have been driven using autonomous driving. For each new navigation goal, save a json file in a way that it could be used in the future to visualize driven paths. Json file should contain the path the robot has driven, status if the goal was reached and total driven distance. Path can be saved as coordinate points which the robot has visited, taken between small time intervals, for example each 0.5 seconds. New file should be saved for each new navigation goal.

<u>Steps in solving task 1:</u>
This task is solved with 2 Nodes in python scripts named “movement_detector_mpo.py” and “store_json.py” . To calculate the distance robot is travelling “movement_detector_mpo.py” is used. This node subscribed to topic /odom which is published continuously in from the simulator both in moving and idle state of the robot. I calculated the difference between every consecutive odometry values to find out Euclidian distance and added that up with previous distance measurement. On a side note, there are always some very small change in the odometry values, so I used a tolerance level of 0.001 to calculate the distance. “movement_detector_mpo.py” is also used for task 2. The “store_json.py” is used to perform the other part of task 1. This script subscribe to both /odom (from simulator) and /moved_distance (published from movement_detector_mpo.py). The odometry value at self.current_odom gets constantly update via /odom topic but added to self.data_obj dictionary only when robot moves known via /moved_distance subscription. This file dumps the data as json only if keyboard interrupt stops the program using ctrl+c. name of the created file is allodom.json. PLEASE, WAIT A FEW MOMENT FOR THE JSON FILE TO BE CREATED.

<u>Room for improvement:</u> I used print() method to output the values on terminal. Logging also works for example: self.get_logger().info() inherited fromROS2 “Node” is tried in the program and exists is the code as commented out section. Other advanced logging and data recording options can be used too.

***One alternative method is to record data from a topic as ROSBAG and later use the python bagpy library to extract the data and save as json.

<u>Steps in solving task 2:</u>
This task is solved with 2 Nodes in python scripts named “movement_detector_mpo.py” and “store_goal_waypoint.py” . To calculate the distance robot is travelling “movement_detector_mpo.py” is used. There are subscribers to following topics:

/waypoints, /odom, /odom_at_interval, /plan, and /moved_distance.

I am taking odometry values every 0.2 seconds(suggested: 0.5 in task description). To achieve this control over time. I created a subscriber to original /odom topic which constantly updates is real time in a very fast rate. The values from /odom topic is simply published via self.publisher_odom_ on /odom_at_interval topic using self.publish_odom_timer_ with a rate defined in timer_period variable.

With every new navigation goal selected a message received in /waypoints topic and calls the method subscribe_message() which then calls the if_goal_changed() to confirm the change of event. If goal_changed is true populate_Json() is called which takes care of rest of the process from recording odom, travel distance, check if goal reached and finally dumping stored data in json file. This multiple layer of checking worked better for me to ensure security to avoid conflict. If the robot stops after a travel or fails to move at all idle_odom value count is incremented. After count reaches limit than it signifies end of process for this specific instance of goal selection and data is dumped in a new json file. With new goal a plan is also received which is filled with way points. The last value in that list is the goal point which is used to calculate if goal is reached. PLEASE, WAIT A FEW MOMENT FOR THE JSON FILE TO BE CREATED. Give a new goal after json file creation is completed.

Room for improvement: I used mostly co-ordinate and in some cases twist for the calculation. Accuracy can be improved by using more parameters. I used only ros2 topic for the tasks which can be improved in some cases with ros2 service.

Some concerns:

1. During the simulation run a list of topics were available but most of those either didn't have relevant data or completely empty. For example: I expected to get data from /goal_pose and /initialpose but those were empty. This problems could be caused either by my own mistake while installing the neo robot libraries or a fault in the system. However, I tried to make a way around those.
2. I tried to use python multiprocessing with a new process id for each new goal but abandoned because the system was hanging frequently. Also in my current version of code it takes little bit of time to create json file. Also accuracy varies with the tolerance levels and timer values I used.
3. Gazebo and rviz was crashing very frequently usually after 1 or 2 new goals. This could probably be prevented by changing some parameters to run gazebo and rviz with lighter loads.

However, points 2 and 3 are more likely caused by my 5 years old laptop struggling to run all the simulation nodes. I'll find that out for sure with more experiments on this system to improve solution for this task. Moreover, to relieve runtime load on the system, I recorded only co-ordinate values of x and y to save in json file but for each necessary topic more details of the relevant parameters can be recorded.