1 Turtlebot Lab

The aim of this lab is to prepare a ROS package for localization, with some additional tools, which will serve as the starting point for your Hands-on On Projects.
This exercise is very important, not only for the localization project, but for all the other projects: For instance, for control and planning, there is a need to know the current robot pose. Therefore, it is essential that you have a basic, yet effective localization algorithm that you can use for two purposes:

- As a base code to extend your Hands-On Localization project.
- As a temporary localization algorithm to test other projects in parallel while you develop the localization project.

2 Installation

2.1 Install the simulation environment

The UdG turtlebot simulation is based on the Stonefish simulator. You need to set-up:

- The Stonefish Library (https://github.com/patrykcieslak/stonefish)
- The Stonefish ROS package (https://github.com/patrykcieslak/stonefish_ros)
- The turtlebot packages (https://bitbucket.org/udg_cirs/turtlebot_simulation)

The turtlebot simulation offers several launch files, where you can decide to run the whole turtlebot, only the kobuki base, or only the manipulator with a fixed base.

2.2 Docker alternative

Alternatively, you can use a docker with all set up.

```
#!/bin/bash
docker pull rogerpi/noetic-stonefish-turtlebot-udg
```

The docker alternative is recommended if you run a ubuntu version $>$ 20.04, where *ROS 1* is not supported anymore. More detailed instructions of installing docker can be found in the attached pdf instructions.

3 Exercises

1. Create a ROS package for localization.
2. Create a node to perform Dead Reckoning based on the wheels' encoders. The sensed wheel velocities are reported in the topic **/turtlebot/joint_states**. Note that you need to check the field **name** as each wheel is reported independently. Moreover, the topic also contains information about the manipulator's joints.

3. Add orientation measurements from the IMU. The topic, located at **/turtlebot/kobuki/sensors/imu_data**, contains orientation, based on magnetometers, and angular velocity, based on gyroscopes. It is recommended to update the filter based on the magnetometer (orientation).

4. Make another node, to control the robot wheels' velocities based on a twist (linear and angular velocity of the robot base). You can then control it not only by publishing manually to the topic but with graphical tools such as **rqt_robot_steering**.

5. Make a plot, using Rviz, of the lidar scans projected on the world frame, while the robot is teleoperated.

6. Two of the most problematic sources of errors in the localization, are due to a miss-calibrated wheelbase distance, and wheel radius. Fortunately, in simulation, these values are perfectly accurate. Test the influence of errors in these values by adding some bias to them, and compare the output of the integration of the lidar with and without orientation updates.

It is important to note that the real robot will report the velocity of the two wheels in a single JointState message, while the simulation does it independently. It is worth finding a solution that fits both cases.