ROS 2 – Aquabot project

1 Content of this project

This project is built around the Aquabot Challenge by Sirehna, where an autonomous boat has to navigate through a windturbine farm and some rocks.
The goals may be various but typically involve performing path planning and path tracking for the boat.
Compared to the initial challenge, all timers have been heavily increased so that users can try various missions in the simulator.

2 Main simulation

The simulation is run with:

```
ros2 launch aquabot_gz turbines_launch.py world:={easy,medium,hard} (default easy)
```

This will run Gazebo with an autonomous boat with two steerable propellers and a pan camera. This boat has to inspect various wind turbines in a rocky region. Turbines have a QR code on one of their side. The robot is available in the `aquabot_description` package.
Several topics, for sensing and action, are available in the `/aquabot` namespace:

Sensor topics

Topic	Message	Description
sensors/cameras/main_camera_sensor/image_raw	sensor_msgs/Image	simulated image
sensors/imu/imu/data	sensor_msgs/Imu	Boat IMU
sensors/gps/gps/fix	sensor_msgs/NavSatFix	Boat GPS
ais_sensor/windturbines_positions	geometry_msgs/Pose	GPS (lat-long) coord of the turbines
sensors/acoustics/receiver/range	std_msgs/Float64	Range to pinger
sensors/acoustics/receiver/bearing	std_msgs/Float64	Bearing to pinger

Actuation topics

Topic	Message	Description
camera/cmd_pos	std_msgs/Float64	Camera angle
thrusters/left/cmd_pos	std_msgs/Float64	Left thruster angle (+- $\pi/4$)
thrusters/right/cmd_pos	std_msgs/Float64	Right thruster angle (+- $\pi/4$)
thrusters/left/thrust	std_msgs/Float64	Left thruster thrust (+- 5000)
thrusters/right/thrust	std_msgs/Float64	Right thruster thrust (+- 5000)

3 Helper packages

3.1 `aquabot_ekf`

The `aquabot_ekf` package contains a node (gz2ros) that handles all sensors except images. Its main use is to convert raw data from Gazebo (GPS, imu, bearings) to metric ones and adding potential noise, ready to be fused by the EKF.
It suscribes to: gps_fix, imu_raw, and the pinger and windturbines position topics.
It publishes:

- gps_xy (geometry_msgs/PoseStamped): metric position based on the GPS info
- imu (sensor_msgs/Imu): IMU plus its covariance
- /aquabot/turbines (geometry_msgs/PoseArray): metric position of the windturbines
- The pinger position on /tf ("pinger" frame)

A configuration file to be used with robot_localization is also available, that can easily consume the topics published by gz2ros in order to get the robot estimate on /tf and odom.
This package is meant *not* to be ready-to-use, a launch file has to be written in order to use the node and EKF according to the available topics.

3.2 `aquabot_motion`

The `aquabot_motion` package contains two nodes to provide some tools for people not interested in this bits.

- Control node (cmd.py): This node subscribes to `cmd_vel` (geometry_msgs/Twist) and odom (nav_msgs/Odometry). It uses a simple PID to control the thrusters of the boat in order to track the desired twist.
- Planning node (planner.py): This node offers a nav_msgs/GetPlan service (`/aquabot/get_plan`) that finds a path from two start and goal poses in the world frame. The path is returned through the service and is also published on the `/aquabot/plan` topic for visualization purpose. This node also subscribes to the `goal_pose` topic. In this case it will use the current pose of the boat as the start, the resulting path will be published on plan.

The planner node also publishes markers for the obstacles, to be displayed in RViz.

4 Project guidelines

4.1 Setup the localization and visualize your boat

Write a launch file that runs the `gz2ros` and EKF nodes with the correct topics. This file may also run RViz to show what we know.

4.2 Ensure the boat can move around

Use the `slider_publisher` package on the `props.yaml` file to see that the robot indeed moves. Then, add the control node to your launch file and make sure the boat is not able to track a velocity setpoint.

4.3 Ensure the planner works

Run the planner node and use the `2D Goal Pose` button on RViz to check if the path is published.
Then write a node that subscribes to this plan and generates a velocity reference for the boat. A look-ahead point is usually quite simple to use.

4.4 Write your state machine for the mission

Now that you can move the boat, track paths and get the position of the windturbines, you have to write a node that will make sure the boat performs a small mission around each turbine. Note that you can build a complex path by calling the planner several times and concatenating the generated paths.

4.5 Optional – the initial Aquabot challenge

Turbines all have a QR code.
When read, its content should be published on the `/vrx/windturbinesinspection/windturbine_checkup` topic. When all QR codes have been read then a pinger will be moved near one of the turbines that needs to be inspected. The boat should stabilize for 30 sec in front of the corresponding QR code and then do a round turn of the turbine.