Aerial and underwater drones

AUV simulation and control

1 Content of this lab

The goal of this lab is to control a thruster-propelled Autonomous Underwater Vehicle (AUV). The AUV should be able to follow a trajectory defined with a sequence of waypoints, and estimate its pose from an EKF.

1.1 The simulation

The simulation is run through Gazebo, which handles hydrodynamics, thrusters and sensors. The actual package that you will modify is to be cloned in `~/ros2/src` with:

```
cd ~/ros2/src
git clone https://github.com/oKermorgant/ecn_auv_lab
cd ecn_auv_lab
ros2ws && colbuild -t # compile this
ideconf # configure IDE for ROS 2 <- do not forget this one
```

This package has a classical ROS structure:

```
ecn_auv_lab
 ├── launch
 ├── params
 ├── src
 ├── urdf
 ├── package.xml
 └── CMakeLists.txt
```

Figure 1: Files used by the simulator

QtCreator can be automatically configured by calling `ideconf` from the `ecn_auv_lab` package. The simulation is run with:

```
ros2 launch ecn_auv_lab sim_launch.py
```

The main option is `gt:=false`. This will prevent the simulator to output the ground truth and force the control to rely on sensor feedback. This will be used in the second part of the lab.

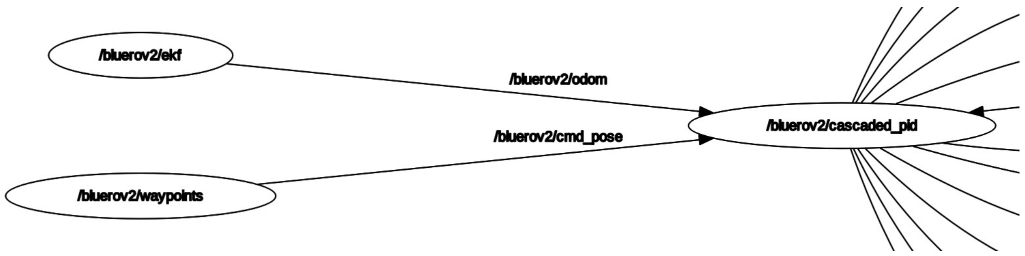
2 Waypoint following

The first task is to finish the waypoint following node defined in `waypoint.cpp`. This node already loads the waypoints defined in `waypoints.yaml` but does not follow them yet. These waypoints should be modified according to the trajectory you want the robot to follow. A waypoint is defined by its (x, y, z, θ) values where θ is the yaw angle. Look at the current code to see how to change a yaw angle to a quaternion needed by the `PoseStamped` message.

Topics can be renamed or the node can be run through a launch file in order to be in the `/bluerov2` namespace. The low-level control (e.g. PIDs) is run through:

```
ros2 launch ecn_auv_lab control_launch.py
```

The `manual` option (default True) will show sliders to give the pose setpoint. When testing your own code, run the control launch file with `manual:=false`. This leads to the following graph where `/bluerov2/waypoints` is your node.



Two GUI are launched:

- Gazebo is the simulator
- RViz is a visualizer and displays the current pose estimate (red) and the current waypoint to be followed (3D frame)

Manually defined setpoints can be easily published by using `manual:=true` (default).

The goal is to modify the source and waypoint list in order to loop through the waypoints (go back to waypoint 0 after the last one was reached). The structure of the `setpoint` message can be found with:

```
ros2 interface show geometry_msgs/msg/PoseStamped
```

which gives:

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

The header is already written but you have to define the pose section from the reading of the loaded waypoint file.

Remember that the AUV may not reach perfectly the desired position, so you should cycle to the next waypoint when the position error is below a given threshold. Two thresholds are loaded from the YAML file: `position_thr` and `orientation_thr`, that may be tuned accordingly to the desired accuracy.
The current position of the AUV can be found on the topic `/bluerov2/state` and is of type `nav_msgs/Odometry` with a structure similar to `geometry_msgs/PoseStamped`, and where we are interested only in the `pose` section.

Once the waypoint node works, uncomment the corresponding line in `control_launch.py` so this launch file runs it when `manual:=false`.

3 Pose estimation from Extended Kalman Filter

The architecture used until then is quite unrealistic. Indeed, data on the topic `/bluerov2/pose` is the ground truth in terms of absolute position and velocity of the AUV (it is coming directly from the simulator). In this section we will replace it by an EKF estimator and see the problems that arise.
Here, Gazebo is still publishing the ground truth but on the `/bluerov2/pose_gt` topic. This data is processed (noise is added) and republished from two simulated sensors:

1. A depth sensor publishing a pose on `/bluerov2/depth` where only the Z-position is meaningful
2. A USBL device publishing a pose on `/bluerov2/usbl` where only the (X,Y,Z) positions are meaningful

Gazebo is also publishing simulated IMU data on the `/bluerov2/lsm` and `/bluerov2/mpu` topics. Finally, RViz displays the estimate of the AUV (in red). For comparison purpose, the ground truth is displayed in blue. The setpoint is still displayed as a 3D frame.
The sensor data (depth, USBL and 2 IMU) should be fused by the EKF node in order to output an estimate of the state on the `/bluerov2/odom` and `/tf` topics that are used by the PID and your waypoint node.
The previous launch files should be run as:

```
# run the simulation without ground truth
ros2 launch ecn_auv_lab sim_launch.py gt:=false
# run the control and track waypoints
ros2 launch ecn_auv_lab control_launch.py manual:=false
```

Then the EKF can be run with:

```
ros2 launch ecn_auv_lab ekf_launch.py
```

The parameter file of the EKF is at `params/ekf.yaml`. It relies on the `robot_localization` package, which offers generic EKF in SE(3). This node creates a 15-components (6 pose, 6 velocities and 3 accelerations) state estimate.

The EKF node can accept any number of input measurements, written as parameters:

- `odom#` for odometry
- `imu#` for imu
- `pose#` for pose

Then the corresponding `<param>_config` parameter is a list of 15 Boolean's indicating which part of the state is to be obtained with this topic.

The parameter file should be modified so that the EKF node subscribes to the available data topics. The Boolean tables should also be updated in order to tell which components of the state are measured by each of the sensors.
You can see the impact of each sensor by running the EKF with:

1. only one or two IMU's
2. IMU's and depth sensor
3. All sensors (IMU, depth, usbl)

You can also change the `params/noise.yaml` file to increase the noise of the simulated sensors and see the consequences in the EKF