Aerial and underwater robots

AUV simulation and control

1 Content of this lab

The goal of this lab is to design a thruster-propelled Autonomous Underwater Vehicle (AUV). After the geometric design, a rough approximation of the hydrodynamic damping will be computed and low-level control (PID's) will be tuned. At the end, the AUV should be able to follow a trajectory defined with a sequence of waypoints.

1.1 Installing the simulator

The simulator is a ROS package and can be installed in the `ros/src` directory with:

```
git clone https://github.com/freefloating-gazebo/freefloating_gazebo.git
```

The actual package that you will modify is downloaded with:

```
git clone https://github.com/oKermorgant/ecn_rasom.git
```

This package has the classical ROS structure:

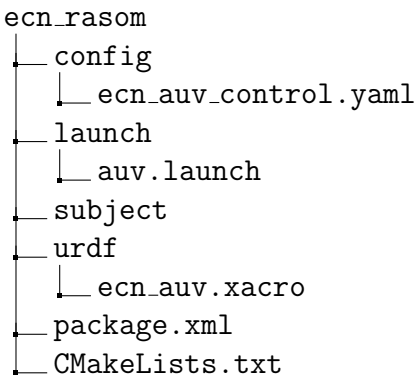

Figure 1: Files used by the simulator

The files to be modified are:

1. `ecn_auv.xacro`: defines the geometry of the AUV
2. `ecn_auv_control.yaml`: PID gain tuning

You will also have to write a .cpp file for the waypoint following, and compile it by updating `CMakeLists.txt`.

2 Definition of a robot in `xacro` format

2.1 Basics of URDF

The `xacro` format allows using variables to generate the URDF model of a robot. In this format, the robot is composed of links and joints. In our case we will assume that the AUV has only one link and no joints.

In the example the AUV is composed of one cylinder, which is described under the `visual` tag. A tutorial is available at `http://wiki.ros.org/urdf/Tutorials/Building a Visual Robot Model with URDF from Scratch`

In addition to the main body, thruster links can be defined with the `xacro:thruster_link` tag, as shown in the example. The xyz position and the roll-pitch-yaw orientation between the main body and the thruster has to be indicated.

2.2 AUV tags

The URDF format was designed to define ground robots. In order to define a AUV, a gazebo plugin is used and will simulate thruster propulsion and hydrodynamic damping. To simulate your own AUV design, update the `thruster` tags in order to give the map that corresponds to your AUV.
At the end of the file we can also find a `buoyancy` tag that indicates the hydrodynamic properties of the robot body. The `origin` tag is used to set the center of buoyancy, that is usually above the center of gravity.

The hydrodynamic damping is defined with 6 coefficients that correspond to the moving directions. In order to have damping values, we will make an assumption on the maximum velocity in each direction (that is a few m/s or 10 deg/s). On each axis, this velocity limit is linked to the maximum force (or torque) and the corresponding damping coefficient with the following relation:

$$m\dot{v} = 0 = \tau_{\max} - k.v_{\lim}$$

where $\tau_{\max}$ is the maximum effort or torque in this direction and depends on the chosen thruster positions.

3 Low-level tuning

Now that the URDF is written, we can use the launch file to run the simulation:

```
roslaunch ecn_rasom auv.launch
```

The simulated AUV is waiting for data on the `body_command` topic, which is computed from the `body_setpoint` topic through PID's.
A generic tool interface in ROS can be run with:

```
rosrun rqt_gui rqt_gui
```

Use the topic publisher plugin to send a position setpoint to your AUV. The Dynamic Reconfigure plugin can be used to change the PID gains without exiting the simulator. Once acceptable values have been found for the gains, you may update the `ecn_auv_control.yaml` file to set the default values.

4 Waypoint following

As the AUV is now capable of reaching a desired position, write a C++ program that makes the AUV cycle through several waypoints.
The message to publish is of type `geometry_msgs/PoseStamped` and has to be published on the topic `/ecn_auv/body_position_setpoint`.

The structure of such a message can be found through `rqt_gui` or with:

```
rosmsg show geometry_msgs/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
```

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. The current position of the AUV can be found on the topic `/ecn_auv/state` and is of type `nav_msgs/Odometry` with a structure similar to `geometry_msgs/PoseStamped`, and where we are also interested only in the `pose` section.

5 Pose estimation from Extended Kalman Filter

The overall architecture developped before is quite unrealistic. Indeed, data on the topic `/ecn_auv/state` is the ground truth in terms of absolute position and velocity of the AUV. In this section we will replace it by an EKF estimator and see the problems that arise.

5.1 EKF package

Download the EKF package from Charles River Analytics:

```
cd ros/src
git clone https://github.com/cra-ros-pkg/robot_localization.git
```

If needed the packages should also be updated. You may save your AUV model before as it may be erased.

```
cd freefloating_gazebo
git pull
cd ../ecn_rasom
git pull
```