# COMP 417/765 Assignment 2

Due: April 10th, 2016
Last revised: March 24th

## 1 Overview

You will implement a vision-based localization method suitable for a swimming robot traveling above a textured planar "ocean bottom", replicated artificially in Gazebo. The geometry and task are depicted in Figure 1.

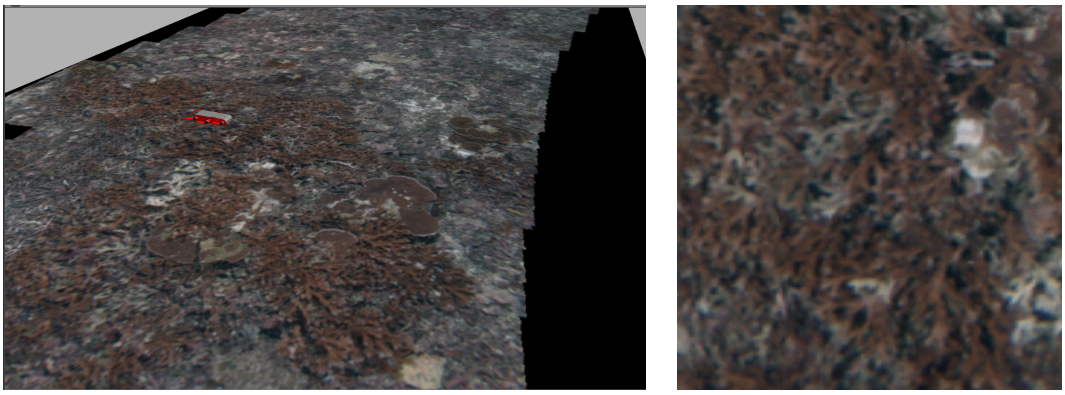

Figure 1: We simulate Aqua swimming over a plane textured with a realistic coral image (right). The simulator gives you the robot's current image (left), which is a view on the map through the geometry of the robot's camera.

The reef image *fishermans_small.png*, which is provided, serves as the map. Your goal is to determine where the robot is in that map as well as possible. You are free to implement any localization algorithm, but we recommend Markov Localization or a Particle Filter, as were discussed in class. COMP 417 students must:

1. Accurately track the robot's location over time when starting from a known pose and when the robot swims only in a planar fashion (that is stays at a fixed depth and does not roll or pitch – state space is x, y, $\theta$).
2. Solve the kidnapped robot problem (starting from unkonwn location),

such that your estimate converges in less than 30 seconds of swimming, again with planar motions and fixed-depth.

COMP 765 students do all of the above but must also handle at least some amount of 3D motion, such as pitch/depth changes during the tracking problem. Ideal solutions handle full 3D motions including roll and pitch, even in the kidnapped robot scenario.

# 2 Getting Started

The provided code archive contains a package *comp*417_*assign*2, which contains a stub localizer node, *localizer_node.cpp*, which currently does a very bad job, purely using Aqua's position commands. You should implement a better version, which can be done by editting our file directly (good to get a fast-start), or by creating your own node from scratch (either C++ or Python are fine). Run with

```
$ rosrun comp417_assign2 localizer_node <path_to_image_map>
```

Your current best-guess esitmate is visualized on the ROS topic

```
/assign2/localization_result_image
```

You may of course also publish additional images on different topics, for exmaple a debug view showing each one of your particles, if you find this helpful.

To evaluate your results, that package also contains *ground_truth_publisher.cpp*, a sample ROS node that displays the path the robot truly followed (obtained by using Gazebo's published state) on top of the map image. This is the ideal output that you should attempt to match as closely as possible with your localizer. Run with

```
$ rosrun comp417_assign2 ground_truth_publisher <path_to_image_map>
```

The ground truth is visualized on the ROS topic /*assign*2/*ground_truth_image*.

There are two ways to test your code for this assignment:

1. "Play-back" simulated robot runs that we have saved for you and made available at:

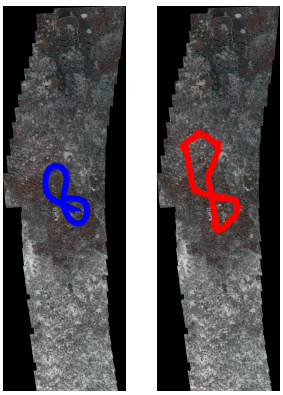

Figure 2: The ground truth result in blue (left) differs from the provided localizer's result in red (right) because the localizer is only using the robot's noisy actions currently. Your code must do better than this.

`http://www.cim.mcgill.ca/~dmeger/comp417_assign2_bags/`

Use the command:

```
$ rosbag play <bag_path> --clock
```

This loads all necessary data from disk and publishes it through ROS just as it was when the simulator was running.

2. You can also compile the code and run the simulator yourself (this will only work on ROS indigo and the particular version of Gazebo that exists on the Trottier lab machines). This option allows you to create any new test situations that you please and to debug thoroughly. Detailed instructions for this process are in the README at the top level of the code archive.

# 3 Environment Description

The visual environment is made up of stitched images from a location known as Fisherman's Reef near the west coast of Barbados. While the true reef is a complicated 3D structure, we have greatly simplified the vision aspect by rendering the stitched image as a perfect plane located at exactly 7 m depth below the simulated water surface. The map image, which is 800x2520 pixels

is represented in the 3D world 16x50.4 metre rectangle, with the coordinate origin $(0, 0)$ in the centre of the image (corresponds to pixel $(400, 1260)$. This allows you to determine the exact 3D coordinate of every image pixel. Also, the simulation process introduces very little noise in the robot's camera, so you will find direct matching of image values between the simulated robot's view and the map works quite well (this is never the case on a real robot, but it makes this assignment more doable in the time-frame).

# 4 Robot Geometry

The robot swims with 6 flippers and the simulator semi-realistically models complex drag and hydrodynamics of this motion. So, while you have access to its desired motions, the real trajectory can be significantly different, and your localizer must make up for this.

In planar mode, the robot is always at a constant 5 m depth (2 m from the map plane). In 3D mode the simulator also starts at 5 m, but the depth can varry between 0 m (hits the water surface) and 7 m (hits the reef).

Figure 3 shows the layout of the camera relative to the robot's frame. The robot rotates roughly around the centre of its shell, which we define as *aqua_base*. The camera is offset at the back-bottom of the shell. The camera's origin defined in the *aqua_base* frame is at $-0.32$ X, 0.0 Y and $-0.06$ Z, all expressed in metres. The camera's optical axis points directly downwards, with the top of the image further "forward" relative to the robot's typical direction of travel.

The simulator forms images by projecting the 3D world through a pinhole camera model. You can access the camera's parameters on the ROS topic:

`/aqua/back_down/camera_info`

Use the $P$ matrix you find there to implement the $x = PX$ projection, remembering that all the 3D points in this world have Z-coord exactly 7 m due to the simple construction.

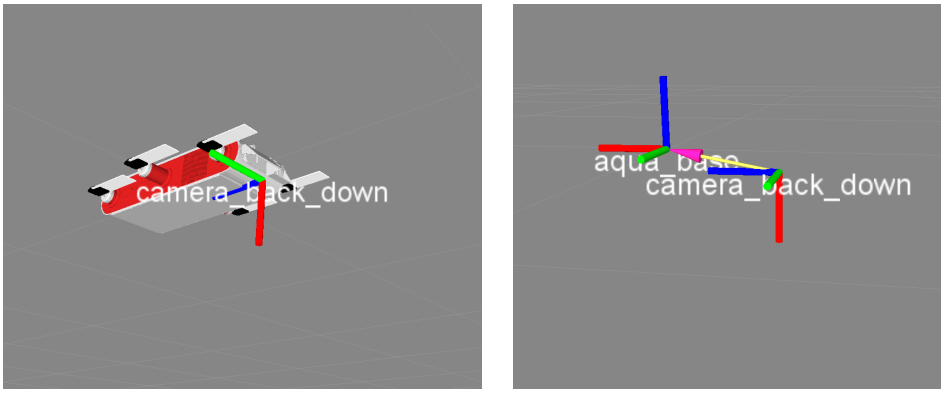


Figure 3: The camera is placed at the back-bottom of the robot's shell. It does not coincide with the centre of rotation, which you may need to account for in your method.

# 5 Hints and Helpful Commands

A recommended first step in coding your localizer is to use the environment and robot geometry we have just described to be able to predict the color of the central pixel that the robot will see from any given pose. This pixel has simple geometry, since it's the one that will point directly downwards from the camera (you can ignore the K matrix here). To validate your result, we included samples from several poses computed by the (e.g., *image_at_0_0_−5.png* was taken as X=0, Y=0, Z=-5). This step, plus a simple per-pixel color comparison will give a great start to form an observation model.

- To access or change the value of a pixel in a cv::Mat image format. image.at¡cv::Vec3b¿(y,x)[0].
- To view the robot's sensed image, and the ground truth and localization outputs, use the image viewer built-in to ROS: *rqt_image_view*
- To set the robot back to its origin (good place to start the tracking problem) use the Gazebo GUI and select Edit menu, Reset Model States.

# 6 Submission

Submit a short (less than one page) write-up of the method you implemented, as well as your code on My Courses.

Our assignment deadline policy for this course is that if you attempt to submit code which is minimally functional by the deadline, you are allowed to consult with other students, the TA or the instructors in order to gain an understanding of what you missed at first. You can submit an improved solution within one week of receiving your initial mark. For this assignment, minimal functionality means your code must be a proper ROS node that compiles and runs, and it must provide noticably better localization than the provided sample localizer node on at least one of the bag files. Failure to meet this criteria, as judged by the TA or failure to submit by the initial deadline removes the chance for re-submission.