Robotic Systems Programming - BOT
Practical Session 3 (TP-3)

Estimating a 6DOF pose using the RGB-D sensor

Panagiotis PAPADAKIS

1 Introduction

The objective of TP-3 is to see in practice a **state estimation** technique within a concrete example, using the simulated TurtleBot2 robot in Gazebo. In particular, you are asked to use the RGB-D (color and depth) sensor of the robot and the **Least-Squares** (LS) state estimation method in order to estimate the 6DOF pose of a color-coded corner in the robot's environment.

Estimating the 6DOF pose of an environment feature could be useful in different contexts, for example:

- **In extrinsic calibration**, when two sensors estimate the 6DOF pose of the same feature with respect to each sensor's frame respectively, which can then be used to resolve the transformation between the 2 sensors themselves, using *triangulation*.
- In **localizing** a robot with respect to a known landmark in the environment

Preliminaries The 6DOF feature is represented as a set of three orthogonal, 3D planes, that are color-coded by Red, Green and Blue wherein the green corresponds to the ground plane and the other two to the sides of the cubes. Each 3D plane is associated to 1 DOF, namely, to 1 of the three orthonormal vectors of the feature's frame and all three planes determine the 3D rotation part of the feature's pose. The remaining 3 DOFs corresponding to the 3D position of the feature are obtained by the point of intersection of the three 3D planes.

A snapshot of the environment feature that the robot is asked to estimate its 6DOF pose with respect to the frame of the robot's sensor is shown in Figure 1.

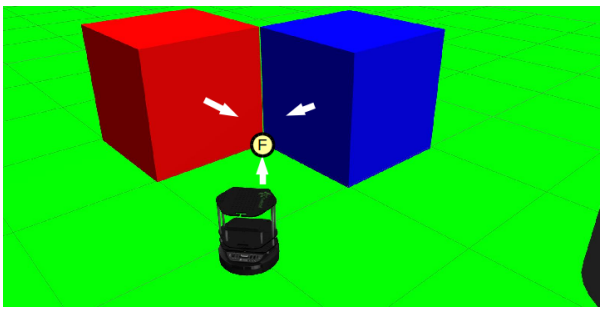
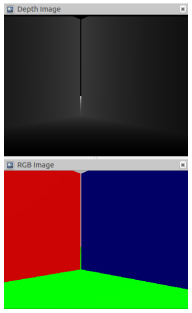
(a)  (b) 

Figure 1: ROS visualization of the problem; (a) orbital view of the TurtleBot2 in proximity to the feature and looking towards the feature and (b) the corresponding depth and RGB images acquired by the RGB-sensor of the robot.

An arbitrary 3D plane can be expressed as follows :

$$\begin{aligned} & a_1x + a_2y + a_3z + a_4 = 0 \\ \Leftrightarrow\ & \tfrac{a_1}{a_4}x + \tfrac{a_1}{a_4}y + \tfrac{a_1}{a_4}z + 1 = 0 \\ \Leftrightarrow\ & ax + by + cz = -1 \end{aligned} \tag{1}$$

wherein the coefficients a, b and c define a vector that corresponds to the orientation of the plane[1], as shown in Figure 2.

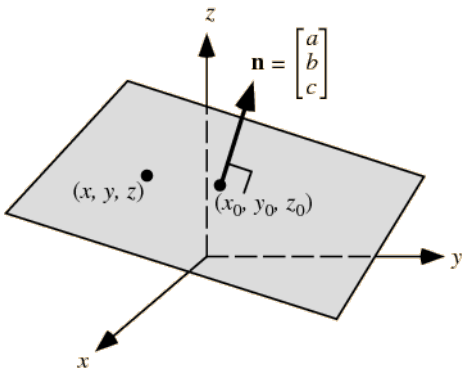
(a) 

Figure 2: An arbitrary 3D plane.

The intersection of three (non-parallel) planes occurs at a single point that satisfies all three plane equations (i.e. it is obtained by solving a 3×3 linear system of equations).

[1] **ATTENTION**: this vector is not necessarily a unit vector.

2 Exercises

Download the rviz configuration file `https://bit.ly/3bvKWUO`, the Gazebo simulated world `https://bit.ly/2SpNpsp`, the turtlebot launch file `https://bit.ly/2UPxGVk` and the partial ROS python code script `https://bit.ly/37mY4Iv` needed for the TP-3. Copy the .world file and the .launch file into the `"worlds"` and `"launch"` folder, respectively, of the package `turtlebot_gazebo`.

Launch the packages need for TP3 by:

```
% roscore

% roslaunch turtlebot_gazebo turtlebot_6DOF_estimation.launch

% rosrun rviz rviz -d tp-3.rviz[2]
```

You can use the ROS package that you created during TP-2[3] to move the robot in a position similar to the one shown in Figure 1, i.e. where it can clearly see the ground plane and **only 1 of the side of the red and 1 side of the blue cube**. Otherwise (if more than one red face or more than one blue face), your ROS package should understand that a 6DOF pose should not be estimated because it will be erroneous.

2.1 6DOF pose estimation

Use the partial ROS python code that you downloaded earlier to realize the TP. You are asked to employ the Least-Squares estimation method described in CS-2 `https://moodle.imt-atlantique.fr/mod/resource/view.php?id=14754` to the problem described earlier. To do so, you will have to create a ROS package (call it `tp3_ros_package`) that will use the 3D point cloud published by the robot in ROS topic:

```
/camera/depth/points
```

which publishes ROS messages of type `sensor_msgs/PointCloud2`[4]. To retrieve these messages, you can use a function of the API of the corresponding python module `sensor_msgs`. The name of the function is:

```
sensor_msgs.point_cloud2.read_points_list()[5]
```

which returns a list of the corresponding points.

[2] assuming that the .rviz file is at the same directory from which you launch rviz

[3] Alternatively, you can use the `turtlebot_teleop` ROS package

[4] You can see all related documentation by typing:
`% rosmsg show -r sensor_msgs/PointCloud2` and
`% rosmsg show -r sensor_msgs/PointField`

[5] Its documentation can be accessed in: `https://bit.ly/2RErNFn`

The known x, y, z coordinates of the points should be used to solve a Least-squares problem for each distinct 3D plane, while the color of the point (either dominant red channel or dominant green channel or dominant blue channel) allows you to know to which 3D plane a point belongs to. To estimate with Least Squares a plane equation, choose 4 or more (e.g. 100) points of the same colored plane that are NOT collinear (you should do this by choosing randomly a subset of points from the total same-coloured points).

Following the earlier plane notation of equation (1), you are recommended to cast the problem of 3D plane estimation as a LS problem as follows :

$$y = H * x \Leftrightarrow \begin{bmatrix} -1 \\ -1 \\ . \\ . \\ . \\ -1 \end{bmatrix} = \begin{bmatrix} x_1, y_1, z_1 \\ x_2, y_2, z_2 \\ . \\ . \\ . \\ x_n, y_n, z_n \end{bmatrix} * \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{2}$$

To perform the necessary matrix operations required for the LS method, you can use the `numpy` python module[6][7].

When you correctly estimate the 6DOF pose then the corresponding coordinate frame should appear in `rviz` at the intersection of the three planes and with axes parallel to their orientation. A demo video file showing the expected behaviour of your project can be accessed in `https://bit.ly/39BRi3n`.

Alongside your code, **you should deliver a similar video recording showing the detection of the 6DOF pose from different nominal and non-nominal viewpoints**. A nominal viewpoint is one where only one red, one blue and one green face is visible from the robot. A non-nominal viewpoint is anything other which should be detected as failure.

[6] https://docs.scipy.org/doc/numpy-1.13.0/reference/index.html
[7] https://docs.scipy.org/doc/numpy/reference/routines.linalg.html