Lab Session 5 – Simulated Turtlebot

In this lab, you will learn about finding the pose of an object in an environment. You will drive the robot in the environment where it will see an image with AR marker on it. The robot's camera will output the distance of this AR marker to the camera. You will use the ROS tf package to transform this pose into a pose in the map.

Set up the singularity environment

Using steps described in lab 1, create a singularity container and then a few Ubuntu terminals to use for later.

Making sure your git clone is up-to-date

Before you do anything else, make sure your git clone of lab5 is up-to-date:

cd $HOME/catkin_ws/src/lab5

git pull

Setting up the Gazebo world

Under lab5/cluedo, you will find the files cluedo_character.zip and cluedo_weapon.zip. Extract these files. They should extract into directories called cluedo_character and cluedo_weapon. Place both under $HOME/.gazebo/models. (In Linux, file and directory names that start with a dot, as in .gazebo, are hidden files/directories. In a file browser you will not see them by default. To see them, hit ctrl-h .)

Now if you look under

$HOME/.gazebo/models/cluedo_character/materials/textures/

you should see an image file Cluedo_character.png, an image of the Cluedo character Colonel Mustard with an AR marker on it. And when you look under

$HOME/.gazebo/models/cluedo_weapon/materials/textures/

you should see an image file Cluedo_weapon.png, an image of the Cluedo weapon Wrench with an AR marker on it.

Start the turtlebot gazebo simulation with the world file lab5.world:

```
export TURTLEBOT_GAZEBO_WORLD_FILE=$HOME/catkin_ws/src/lab5/src/lab5.world

roslaunch turtlebot_gazebo turtlebot_world.launch
```

The simulator should load the same environment from lab4 with a small difference: On the walls, there are the posters of the Cluedo character and weapon. Make sure you see where these posters are.

Start localisation using the launch file from lab4:

```
roslaunch simulated_localisation.launch map_file:= <path-to-my-map-yaml-file>
```

Since the environment is the same as the lab4 environment, you can use the map from lab4.

Finally, start RViz with the navigation interface:

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

Localise your robot using the '2D pose estimate' button in Rviz, as you did in lab4.

In Rviz, on the right pane, under Views, select the view: Orbit (rviz), then zoom in / rotate your viewer to clearly see the robot.

TF frames

In ROS, tf is a package that lets us to keep track of the poses of multiple coordinate frames.

In RViz window, on the left pane, under Displays, enable TF by making sure the checkbox near it is checked. At this point you should be seeing many tf frames appearing in the environment and on the robot. Each tf frame is represented by three red-green-blue axes.

On the left pane, click on the small triangle to the left of TF to extend the submenu. Make sure “Show names” is checked. This should visualise the names of the TF frames in RViz. Similarly, extend the submenu “Frames”. Let’s try to hide most of the TF frames, so that you can see them more clearly. Chech/uncheck “All Enabled” box to hide all the TF frames. Then in the list of frames, check only "base_link", "camera_depth_frame", and "map". Notice that the "map" frame corresponds to the origin of your map, the "base_link" frame is attached to the base of your robot (it may be a bit hard to see because the robot’s body is in the way), and the "camera_depth_frame" is attached to the robot's RGBD camera. All these TF frames are constantly published by different nodes in the ROS environment. For example, localisation specifies where the “base_link” of the robot is in the “map” as the robot moves. The “camera_depth_frame” is known and fixed relative to the “base_link” when the robot is built. There are many frames like this! Check "All Enabled" back again to enable all the frames. These are all the TF frames being published to the ROS environment right now.

Let’s say you have developed a computer vision program to detect objects in a camera image. Using it, let’s say your program detected an object 0.9 m away from the camera. Your program can declare this to the other ROS nodes by publishing a TF frame for the detected object. We can fake the output of such a hypothetical program here by publishing a static frame. In a new terminal window, execute:

```
rosrun tf static_transform_publisher 0.9 0 0 0 0 0 /camera_depth_frame /object 100
```

The above command publishes the “/object” frame to be at x=0.9 with respect to the “/camera_depth_frame”. In RViz you should see a new “object” TF frame appearing 0.9 m in front of the robot camera.

Your program knows where the object is relative to the camera, but a good question to ask to your robot is: What are the coordinates of this object in the room (i.e. map)? You can ask TF this question. Open the file lab5/src/tf_listener.py and inspect it. You will see that we are creating a TransformListener object and then we are using it to lookupTransform of a child_frame in a parent_frame. In our case, since we want to know where “/object” is in “/map”, our parent_frame is “/map” and our child_frame is “/object”. Make this change and execute this script. It will print out the translation and rotation of the “/object” in the ”/map” frame.

But this object was a fake object we published. Stop the “tf static_transform_publisher” to stop this. Let’s really use the robot’s camera to detect an AR marker. Launch the ar_tracking.launch file

```
cd $HOME/catkin_ws/src/lab5/launch

roslaunch ar_tracking.launch
```

This runs a program that detects AR markers in the camera image. This program, when it detects an AR marker, publishes the pose of this marker as a TF frame.

Now, in a separate terminal, run the teleoperation program like you did in lab4:

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

Use the keyboard to drive the robot such that it approaches one of the posters with the AR marker on it. As you do this, keep the RViz window in view. When the AR marker is detected you should see a new TF frame appearing in front of the robot, with the name “ar_marker_0”. When you see this TF frame appear in RViz, stop moving your robot. Now edit the tf_listener.py script to print the position of the ar_marker_0 frame in the map frame, and execute the script. Congratulations, you have found where the character/weapon image is in the map!

Also note that the AR tracker program publishes to a topic called ar_pose_marker. Run:

```
rostopic echo ar_pose_marker
```

If any markers are detected, you will see them appear in the “markers” list in this message. If you want you can also subscribe to this topic to listen to AR marker detections.

Lab Session 5 – Real Turtlebot

Initializing your group's git repo

If you do not remember how to do this, please read the section "Initializing your group's git repo" from the lab1 worksheet, and initialize your group's git repo for lab5.

Setup Turtlebot laptop for your group

If you do not remember how to do this, please read the section "Setup Turtlebot laptop for your group" from the lab1 worksheet.

Start the robot:

```
roslaunch turtlebot_bringup minimal.launch
```

and the astra sensor:

```
roslaunch astra_launch astra.launch
```

For the simulated Turtlebot you used the ar_tracking.launch file to start AR tracking. For the real Turtlebot, we will use a different launch fie:

```
roslaunch ar_track_alvar pr2_indiv.launch
```

Use the instructions from lab4 to map the environment and localise and teleop your robot in this environment. Teleop your robot to a position such that it sees a poster with an AR marker on it.

You can use:

```
rosrun image_view image_view image:=/camera/rgb/image_raw
```

to see what your robot camera is seeing at any moment.

You can use:

```
rostopic echo ar_pose_marker
```

to see if any markers are detected at any moment by the AR tracker.

When an AR marker is detected, stop, and run the tf_listener.py script to print out the position of the marker in the map.

Do not forget to push any change in your code to git! Otherwise you will lose them when you logout of the Turtlebot laptop!

Appendix: Cluedo posters in simulation

As you work on your project, you might want to test your code with different positions of the Cluedo posters. Moving posters is easy in the real environment; you can just stick them on different walls. In simulation, you will need to edit a file.

Let's say you want to change the Cluedo character's position. Open the file: $HOME/.gazebo/models/cluedo_character/materials/model.sdf . In this file on line 23, you will see the pose of the poster specified between <pose> tags. The first three values are x, y, and z, and the next three values are roll, pitch, and yaw. You can change these values and restart turtlebot_gazebo to move the poster in the simulated world.

Similarly you can move the Cluedo weapon's position.

For your project, we provided multiple different character and weapon images on Minerva. In simulation, the default ones we used are Colonel Mustard and the Wrench weapon. However, you can also change these if you want to test with different character/weapon images. To do this, simply overwrite the image file under $HOME/.gazebo/models/cluedo_character/materials/textures with the file you want to use. Make sure to keep the name same though (Cluedo_character.png for the character and Cluedo_weapon.png for the weapon).