PERSONAL HELPER ROBOTIC BUTLER

A.L.F.R.E.D

INTRODUCTION

Basic aim of the project is to design and build a personal robotic helper that will be able to do jobs like getting coffee for the user, for example. The robot needs to be localized in a 2-dimensional map of the working area. The robot needs to autonomously maneuver in the environment to get to the target location to grab the cup of coffee, while avoiding obstacles in the way.

A unified solution to this problem can be given by ROS (Robotic Operating System). ROS is a flexible framework for writing software for robots [ros.org]. It is a set of libraries and algorithms which help us create robust software for robots across platforms. The ROS libraries and tools are open source written in C++, Python and XML under Linux platform. The beauty of ROS is the fact that it is an open source solution to which researchers across the globe can contribute. The main contributors who have contributed the core project libraries can be found on their website [ros.org/contributors].

We are using the kinetic distribution of ROS and the ROSARIA package that helps ROS communicate with the robot, i.e. Pioneer 3DX, on the Ubuntu 16.04 LTS (64-bit) distribution. The installation instructions of various packages are given in the further part of this documentation. The instructions regarding setting up the robot with ROS and ARIA, 2-d mapping of the office area and setting it up for the localization and navigation in that map are also given.

INSTALLATION

1. ROS Kinetic – Follow the official tutorial given in the following link to install ROS kinetic package - http://wiki.ros.org/kinetic/Installation/Ubuntu. Note that kinetic version of ROS is recommended for the latest Ubuntu distribution.

2. ROSARIA – Follow the official tutorial in the following link to install ROSARIA package compatible with the kinetic version –
http://wiki.ros.org/ROSARIA/Tutorials/How%20to%20use%20ROSARIA.
During this process, if we get an error during de-packaging of the aria libraries, change directories to /usr/local/Aria and rebuild Aria by the following commands:
> make clean
> make -j4

3. Joystick mode – The joystick package in ROS can be installed directly from Ubuntu using the apt-get package.
> sudo apt-get install ros-<distribution>-joy*

4. Gmapping – The gmapping package and slam-gmapping package can be installed using apt-get.

> sudo apt-get install ros-<distribution>-gmapping

> sudo apt-get install ros-<distribution>-slam-gmapping

5. AMCL – AMCL is the Adaptive Monte Carlo Localization package that is also installed using apt-get. It is a probabilistic localization system for a robot that moves in 2-dimensional space. It uses adaptive Monte Carlo localization that uses the particle filter to track the pose of the robot against a known map [ros.org/amcl].

> sudo apt-get install ros-<distribution>-navigation*

> sudo apt-get install ros-<distribution>-amcl

6. RVIZ – RVIZ is the visualization tool in ROS that helps us simulate the operations of the robot. It allows us to model the robot using XML descriptions and configurations. We can pull up the map of our working area and localize and navigate the robot in the known map all in simulation.

> sudo apt-get install ros-<distribution>-visualization*

> sudo apt-get install ros-<distribution>-rviz*

ROS - NETWORKING

The networking solution provided by ROS uses the Secure Shell (ssh) protocol to set up communication between different nodes that can be running across different machines. To setup the communication between different machines, we need to setup certain environment variables in each of the machines. Note that all these machines are required to be on the same Local Area Network for this to work. This being Secure Shell protocol, this is not a necessary condition, but the following setup assumes that the machines are on the same LAN.

1. Open the /etc/hosts file using a suitable editor and add the IP addresses of all the machines that need to work in the ROS network.

2. Open the bashrc file using a suitable editor and add he following lines at the end of the file:

> export ROS_HOSTNAME=<host-name of the local machine>

> export ROS_IP=<IP address of the local machine>

> export ROS_MASTER_URI=http://<host-name of the master machine>:11311

(Note that if we don't want to use the ROS networking, we use the host-name of the local machine instead)

3. Either exit out of all the terminals and reopen or source the bashrc file in each terminal for the setup to take effect.

CREATE NEW PACKAGE

Prerequisite – We need to have a catkin workspace created before we can go ahead and create a ROS package. To do this execute the following commands:

> source /opt/ros/<distribution>/setup.bash
> mkdir -p ~/catkin_ws/src
> cd ~/catkin_ws/src
> catkin_init_workspace
> cd ~/catkin_ws
> catkin_make (this command compiles all the packages and the code therein)

Source the new setup.bash file before continuing

> source devel/setup.bash (execute this from the catkin_ws directory)

To make sure everything went fine, echo the ROS_PACKAGE_PATH variable and we should see the path of the created workspace included in the path.

To create a new package in ROS, follow the following steps:

> cd ~/catkin_ws/src
> create_catkin_pkg <package_name> <dependencies>
Example - create_catkin_pkg my_package std_msgs roscpp rospy

> cd ..
> catkin_make
> source devel/setup.bash (execute this from the catkin_ws directory)

We can check or edit the dependencies of the package in the package.xml file that is auto generated in the package directory. We can write the code in the src directory inside our package directory in C++ or Python.

RPLIDAR

RPLIDAR is a low cost LIDAR sensor that is useful for indoor SLAM (Simultaneous Localization and Mapping) application [ros.org/rplidar]. LIDAR is a Light Imaging Detection and Ranging sensor that provides a 360-degree field view, 5.5/10 Hz rotating frequency and a 6-meter range. We need to install the rplidar package to work with this hardware in ROS.

> cd ~/catkin_ws/src
> git clone https://github.com/robopeak/rplidar_ros.git
> catkin_make (from catkin_ws directory)

We also installed the hector slam package that works with the rplidar in order to create maps

> cd ~/catkin_ws/src
> git clone https://github.com/tu-darmstadt-ros-pkg/hector_slam.git
> catkin_make (from catkin_ws directory)

SETTING UP ROBOT FOR MAPPING

We need to set up our robot using transform frames (tf) as follows:

Publish static transforms as –

map => odom
<node pkg="tf" type="static_transform_publisher" name="map_to_odom_broadcaster" args="0 0 0 0 0 0 /map /odom 1000"/>
odom => base_footprint
base_footprint => base_link
base_link => base_stabilized
base_stabilized => base_frame
base_frame => laser
base_frame => nav

Also set the parameters as follows:

<param name= “pub_map_odom_transform” value= “true” />
<param name= “map_frame” value= “map” />
<param name= “base_frame” value= “base_frame” />
<param name= “odom_frame” value= “odom” />

Find the sample “slam.launch” file in our code source in github (_____). Keep this launch file in ~/catkin_ws/src/hector_slam/hector_slam_launch/launch and then source the ~/catkin_ws/devel/setup.bash file.

MAKING OUR OWN MAP

Follow the following steps to make our own map:
(Note that in each terminal we open we need to first source the ~/catkin_ws/devel/setup.bash file)
We also need the rplidar to be connected to the machine

1. roscore

2. roslaunch rplidar_ros rplidar_ros.launch (in another terminal)

3. rosbag record -O <file_name> /base_scan /tf /odom (in another terminal)

4. roslaunch hector_slam_launch slam.launch (in another terminal)

5. Now move around with the rplidar around the working area. We will see the map getting built. Avoid rapid movements of the rplidar. Cover the entire area that is required to be mapped. Avoid leaving any grey areas in the process. Try to come back to the starting position at the end.

6. Now kill the rplidar.launch (ctrl+c) only. DO NOT kill any other launch file.

7. rosparam set /use_sim_time true (in another terminal)

8. rosrun gmapping slam_gmapping scan:=base_scan
   (source /opt/ros/<distribution>/setup.bash first)

9. rosbag play <file_name.bag> (in another terminal)

10. rosrun map_server map_saver (source /opt/ros/<distribution>/setup.bash first)

11. rosparam set /use_sim_time false

A map.pgm and map.yaml file is created in the current directory. Now we can kill all processes in all terminals. (the .pgm is the image file and .yaml is the parameter file of our map).

LOCALIZATION WITH AMCL IN KNOWN MAP

We will now use the map we just created to use the AMCL package for localization. We have already installed this package in the 'installation' section. Do the following in separate terminals:

1. roscore (source ~/catkin_ws/devel/setup.bash first)

2. rosrun map_server map_server <path-to-map.yaml>
   (source /opt/ros/<distribution>/setup.bash first)

3. roslaunch amcl move_base.launch (source /opt/ros/<distribution>/setup.bash first)

4. rosrun rviz rviz

Rviz will open up. Inside Rviz, add the following topics –

1. Map => subscribed to /map

2. Path => subscribed to /move_base/TrajectoryPlannerROS/global_plan

3. Pose => subscribed to /move_base_simple/goal

4. Polygon => subscribed to /move_base/ global_costmap/ obstacle_layer_footprint/ footprint_stamped

Note – Find the move_base.launch file in our code source in github (_____). Keep this file in /opt/ros/<distribution>/share/amcl/examples directory and then source the /opt/ros/<distribution>/setup.bash file.

Now click on '2D pose estimate' button and click on the polygon and set the direction of the arrow saying that the currently our robot is facing this direction.

Now click on the '2D nav goal' button and set a destination and pose on any point on the map. The global path plan is now displayed on the map. We can see that the path from the current position to the nav goal is planned in rviz.

To make the localization, we subscribed to a topic called 'slam_out_pose' which is published by the 'hector_mapping' package. For this purpose, we included the 'hector_mapping'

package and configured it to be subscribing and publishing the correct topics. This package was publishing to the ‘map’ topic, a newly generated map and this was conflicting with the already known map that the map_server was publishing. We configured the ‘hector_mapping’ package to publish is data to a topic that we called ‘testing_map’, thus routing it elsewhere out of the way of the known map. (This change was made in the MappingROS.cpp code in the hector_mapping package) Now, we had pose data and a known map which we combined so as to localize the robot in the map. (The robot was mounted with the rplidar).

The basic mechanism of this localization is that it collects the laser_scan data from the rplidar and compares it with the already saved data (map) to figure out where approximately on the map it is at that point in time. When we combine this data with the odometry data from the robot (RosAria/pose) this localization will get less approximate and more accurate.