A PROJECT REPORT

On

SLAM AND NAVIGATION USING LIDAR FOR A KUKA YOUBOT

By

HARIKRISHNAN SURESH

NITK SURATHKAL

Under the guidance of

Dr.T.ASOKAN

Mr. NAGAMANIKANDAN G.

DEPARTMENT OF ENGINEERING DESIGN

IIT MADRAS

May 2016- July 2016

Acknowledgement

I wish to express my sincere thanks to the people who extended their help during the course of my project.

I express my deepest gratitude to Dr.T.Asokan, Department of Engineering Design, IIT Madras for providing me the opportunity to work under his guidance. I thank him for the support and motivation he offered me throughout this project.

I also record my thanks to Mr. Nagamanikandan G, research scholar, for his constant technical support and guidance which has immensely contributed to the successful completion of this project.

Table of contents

1. Abstract

A mobile robot is an automatic machine capable of performing locomotion. Mobile robotics is a trending research field, owing to its large scale use in industrial and commercial sector. The fundamental issues concerning mobile robots are gathering information about its present location, about its goal location and planning how to reach there. Building maps of unknown environments is a pivotal problem in mobile robotics, as it acts as a pre-requisite to addressing the above issues. Also, these mobile robots must use a Robot arm or manipulator to perform the applications it is built for. Robot manipulators, have evolved from the traditional structure of being bolted down to the assembly line to being mounted on a mobile platform. The flexibility is greatly improved now with the ability to move around and perform the required repetitive tasks wherever it is most effective.

ROS has revolutionized the way robots are programmed for various applications. ROS contains all the functionalities like a normal Operating System. A lot of research institutions have started to develop projects in ROS by adding hardware and sharing their code samples. Also, the companies have started to adapt their products to be used in ROS.

In the field of SLAM and navigation, ROS packages are available and well-documented for a few platforms like the PR2 robot. This project aims at implementing a ROS stack, youbot_navigation stack on a KUKA youBot and preparing a manual containing step-by-step instructions to perform 2D SLAM and autonomous navigation. The youbot_navigation stack was written for ROS Fuerte, so the preliminary work was modifying the packages to run on ROS Hydro.

2. Overview

The first stage of the project was the learning phase - acquiring a basic understanding of ROS, programming in ROS and also basics of mobile robot navigation. The sensor was then interfaced with ROS, calibrated and the sensor data was visualized on the computer. Youbot was setup and programmed to be tele-operated using a keyboard. With the help of sensor data and odometry data, the youbot was programmed to build a 2D Map of the unknown environment, localize itself relative to the map and autonomously navigate to the user-defined location in the map. The next stage of the project was to learn about the path planning algorithm used by the youbot for navigation and optimize it. The last stage of the project was interfacing a dynamixel servo with ROS, use a rotating LIDAR sensor to build a 3D Map of the environment.

The ROS distro used for this project was ROS-Hydro on Ubuntu 12.04 LTS.
Youbot supports ROS-Fuerte and ROS-Hydro.

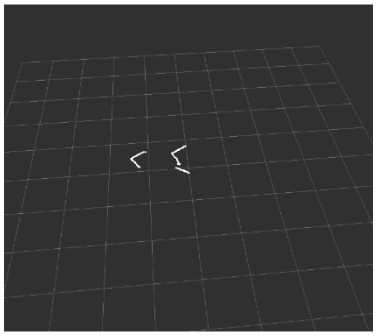
3. The sensor – Hokuyo URG-04lx

3.1. About the sensor

Hokuyo URG-04lx is the sensor used to perform SLAM and navigation. It is a laser sensor for area scanning. The scan area is 240^0 with a maximum radius of 4m. Principle of distance measurement is based on calculation of the phase difference, due to which it is possible to obtain stable measurement with minimum influence from object's color and reflectance.
The sensor gives laser scan data in the form of range and bearing angle.

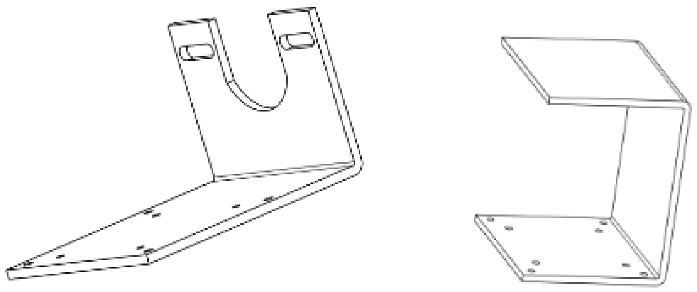
3.2. Interfacing sensor with ROS

The **hokuyo_node** is a ROS node to provide access to SCIP 2.0-compliant Hokuyo laser range finders. The sensor is interfaced with ROS and the laser data is visualized in Rviz. For complete set of tutorials, refer http://wiki.ros.org/hokuyo_node/Tutorials


The sensor (left) and raw sensor data visualized in Rviz(right)

3.3. Sensor mounts

The mounts for attaching sensor and emergency switches to the youBot were designed in CATIA using the generative sheetmetal design tool. The mounts were machined using Aluminium sheets. The CAD drawing of hokuyo sensor was used as reference for deciding the measurements for the mounts.


CAD Drawings of sensor mounts

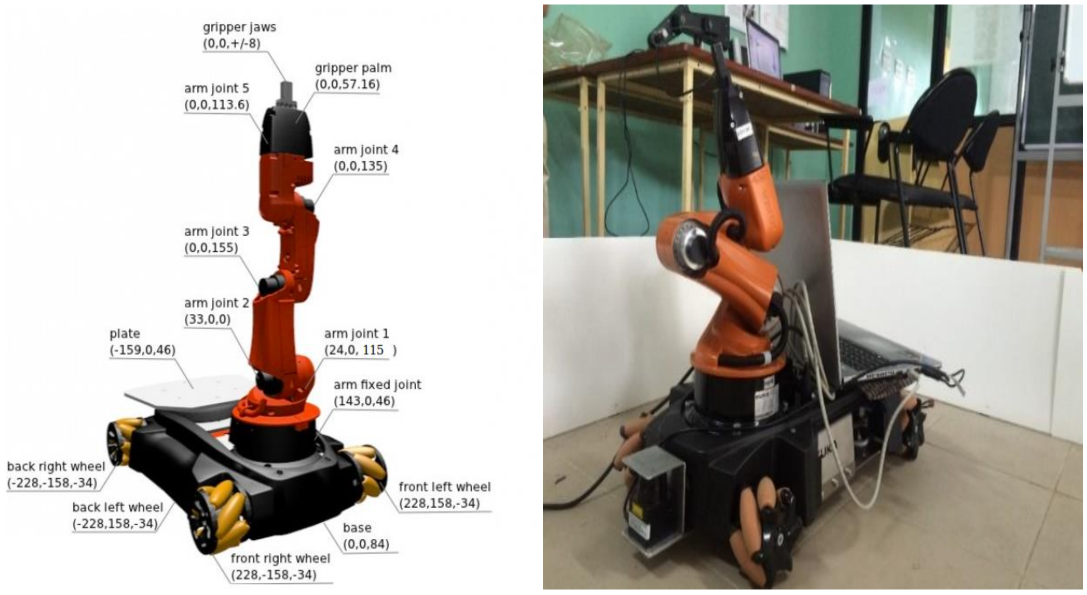
4. The platform – KUKA youBot

4.1. About the platform

KUKA youBot is a mobile manipulator, wherein a 5 axis arm is mounted on a base. It is an omni-directional robot that uses Swedish/Mecanum wheels for motion. It has a detachable type gripper as the end effector. It houses an on-board PC that runs on Ubuntu Linux, with support for ROS Fuerte and ROS Hydro.

4.2. ROS Wrapper for youbot

The ROS wrapper of the **youbot driver** (or KUKA youBot API) provides an interface of the youbot driver to the ROS framework. With this wrapper, the base and the arm of the youBot can be moved by sending ROS messages.



The KUKA youBot

5. The actuator – Dynamixel RX-64

5.1. About the actuator

The Dynamixel series robot actuator is a smart, modular actuator that incorporates a gear reducer, a precision DC motor and a control circuitry with networking functionality, all in a single package. Position and speed can be controlled with a resolution of 1024 steps. Feedback for angular position, angular velocity, and load torque are available.
The actuator used for this project is the **Dynamixel RX-64**. The aim is to attach the LIDAR to the servo and enable a pitching motion for the LIDAR.

5.2. Interfacing actuator with ROS

The servo was made to rotate to the specified angle using ROS messages. A ROS node called **sweep.cpp** was created to perform 270^0 sweep continuously with the servo. The **sweep** node was successfully run on ROS. A dynamic TF publisher was also created which creates a continuously rotating frame that mimics the servo motion.

For further details, refer http://wiki.ros.org/dynamixel_controllers

The actuator

6. 2D Navigation using youBot

6.1. Introduction to mobile robot navigation

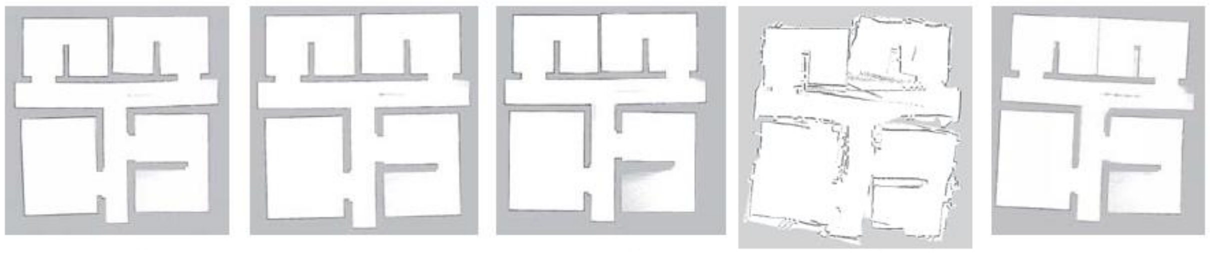
Robot navigation means the robot's ability to determine its own position in its frame of reference and then to plan a path towards some goal location. In order to navigate in its environment, the robot requires representation, i.e. a map of the environment and the ability to interpret that representation.

Hence for proper navigation, the two most important pre-requisites are

1. Mapping - A robot, that has perfect knowledge of its position in an unknown environment, will be able to build a map or floor-plan of the environment by sensing from various positions and integrating these measurements. Map representations can be grid-based, feature-based or topological.
2. Localization - A robot that is provided with a well-defined map of the environment will be able to sense from various positions, relate these measurements to the world model and determine its position and orientation with respect to the map.

However, in situations where neither a map nor any information on the position of the robot is known, the mobile robot will build a map of the unknown environment and simultaneously localize itself relative to this map. This is called Simultaneous Localization and Mapping SLAM.

The 2D SLAM algorithms implemented in ROS are - HectorSLAM, Gmapping, CoreSLAM, KartoSLAM and LagoSLAM. Among these, Gmapping was seen to be the most robust, accurate and computationally least expensive. Gmapping uses both laser scan data and odometry data to construct an occupancy grid map of the environment. A comparison between the different 2D slam techniques can be found in [8], and is also depicted in the image below.



(a) HectorSLAM (b) Gmapping (c) KartoSLAM (d) CoreSLAM (e) LagoSLAM

With a well-defined map and localization information, the robot will be able to navigate to a user-defined location in the map by itself. The robot uses a path planning algorithm to reach the goal through the shortest path and is also capable of avoiding static and dynamic obstacles in its path.

The map can also be extended to 3D using a modified sensor input. A 3D representation of the environment is useful for a mobile manipulator to obtain information about the pose of an object to be manipulated.

6.2. The youbot_navigation stack

This software includes a ROS navigation stack adapted for the KUKA youBot. It provides local navigation and obstacle avoidance functionalities based on the laser scanner readings. It contains all the files needed to perform 2D SLAM and navigation using youBot. youbot_navigation is a stack which contains 3 packages - **youbot_navigation_common, youbot_navigation_global and youbot_navigation_local.**

The **base_front_hokuyo_node.launch** runs the hokuyo node and the static transform publisher between the laser frame /laser and mobile frame /base_footprint. The distance between centre of youBot and centre of laser in the experimental setup was measured and values were entered in the static TF.

bringup_navigation.launch includes the commands to launch hokuyo node, youbot driver and move base.

2dslam.launch contains the commands to run the slam_gmapping node.
GMapping is the most widely used SLAM package in mobile robotics. It is a highly efficient Rao-Blackwellized particle filter to learn grid maps from laser range data. In RBPF, each particle carries an individual map of the environment. RBPF obtains a joint information about the map *m* and the trajectory $x_{1:t}$ using the sensor measurements $z_{1:t}$ and odometry measurements $u_{1:t-1}$.
A detailed description about this algorithm is mentioned in [7].

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t-1}) = p(m \mid x_{1:t}, z_{1:t}) \cdot p(x_{1:t} \mid z_{1:t}, u_{1:t-1}).$$

GMapping generates an occupancy grid map of the environment. The occupancy grid representation employs a multi-dimensional (typically 2D or 3D) tessellation of space into cells, where each cell stores a probabilistic estimate of its state. Cell states have 3 values – Occupied, Empty and Unknown.

amcl.launch contains the commands to launch a previously constructed map and also run amcl node.
AMCL stands for Adaptive Monte Carlo Localization. AMCL is a probabilistic localization system for a robot moving in 2D.
It implements the adaptive (or KLD-sampling) Monte Carlo localization approach, which uses a particle filter to track the pose of a robot against a known map. KLD– sampling is a variant of Monte Carlo Localization where at each iteration, a sample size is calculated.

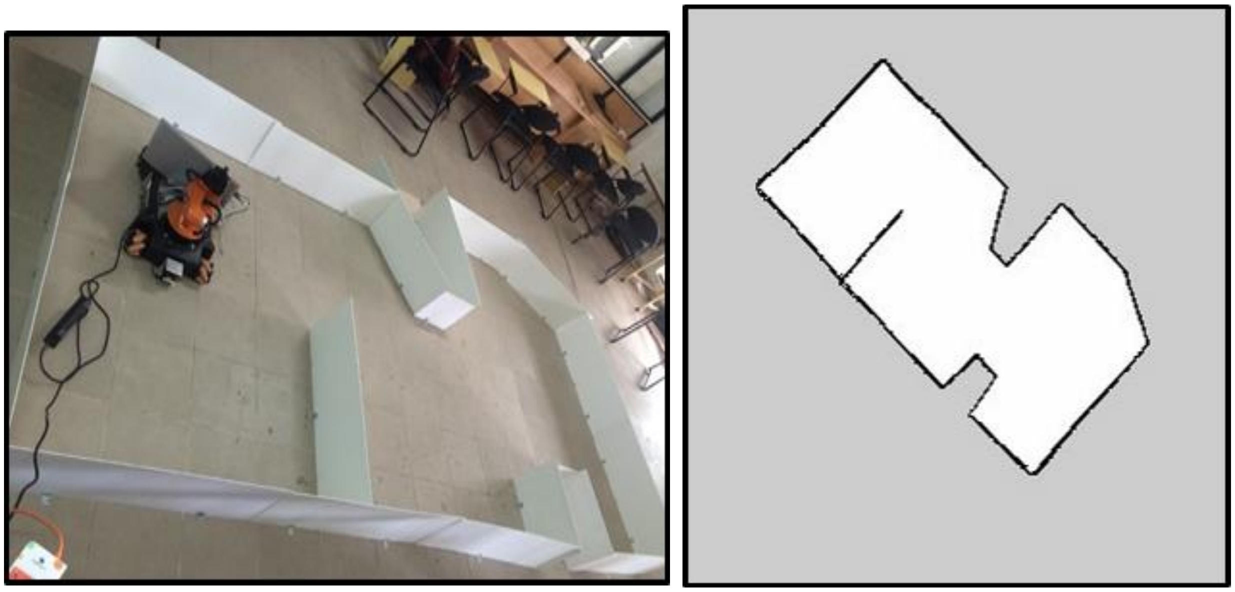
move_base_global.launch includes commands to run the global and local path planners, and their respective cost maps. To use a different path planning algorithm, changes must be made in this launch file.
move_base node uses
1. **navfn** as the global path planner : navfn provides a fast interpolated navigation function that can be used to create plans for a mobile base. The planner assumes a circular robot and operates

on a costmap to find a minimum cost plan from a start point to an end point in a grid. The navigation function is computed with Dijkstra's algorithm

2. **base_local_planner** as the local path planner: It contains a controller that produces velocity commands to send to a mobile base based on a global plan and a costmap.The controller's job is to use this value function to determine dx,dy,dtheta velocities to send to the robot. This package provides implementations of the Trajectory Rollout and Dynamic Window approaches to local robot navigation on a plane. DWA approach is used for this project, as it gives more accurate results when the acceleration limits are high.

6.3. 2D SLAM – Experimental results

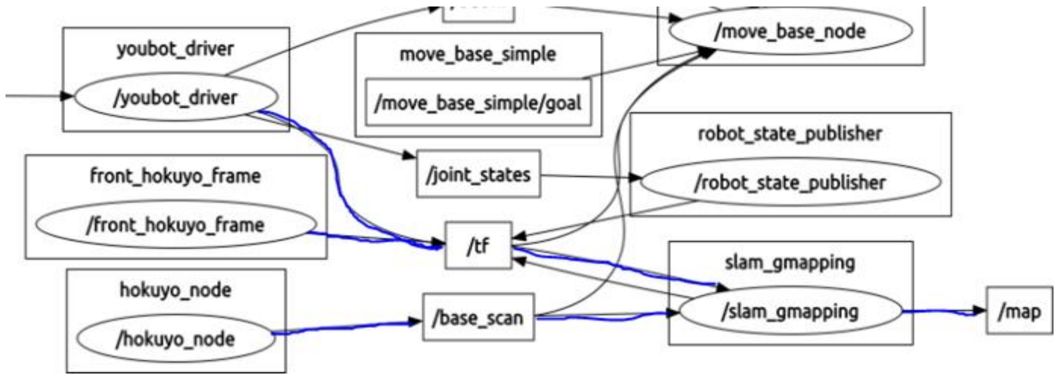
Refer the lab manual for step-by-step instructions on how to perform 2D SLAM using the youBot.


The arena and the corresponding occupancy grid map built using youBot.

The arena was constructed using poly-propylene sheet.
Accuracy of grid maps depend on the extent of perfection of the arena. A well built arena, with straight walls ensure proper reflection of the laser and results in correct sensor data. Bent sheets cause poor sensor data and results in uncertainty in the map.

‘

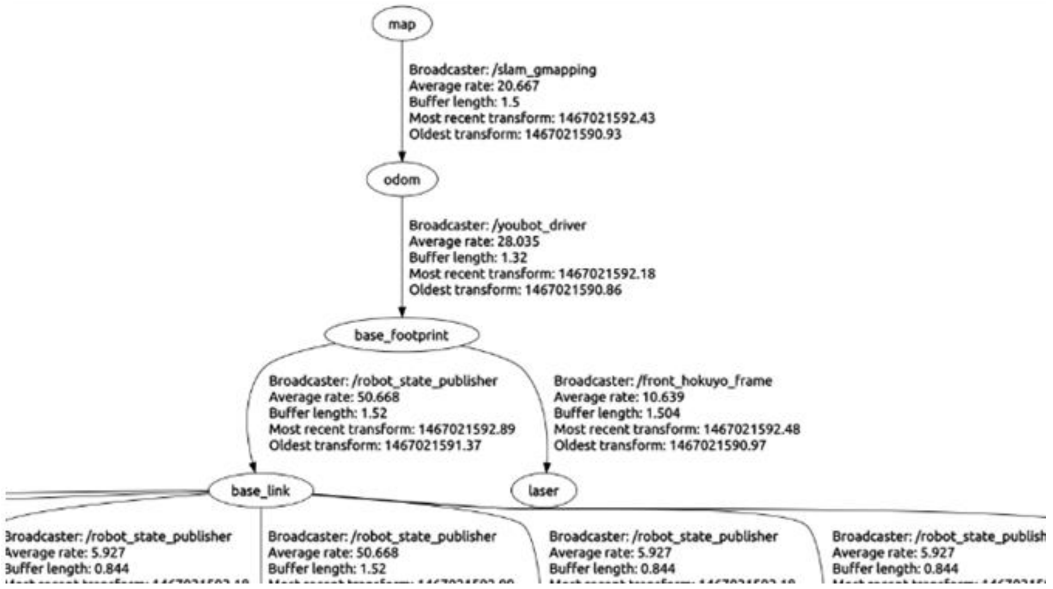
An **rqt_graph** provides a visualization of the ROS computation graph. It contains a list of nodes, topics and how they are related.



Rqt_graph of slam_gmapping

The **slam_gmapping** node takes /base_scan topic and /tf topic as input, and produces /map as output.
/base_scan contains the laser data, published by hokuyo_node.
/tf contains the transforms between fixed frame /odom, mobile frame /base_footprint and laser frame /laser. Youbot_driver publishes the tf between /odom and /base_footprint whereas front_hokuyo_frame node publishes the static tf between /base_footprint and /laser. front_hokuyo_frame was created but youbot_driver comes with the robot.
slam_gmapping publishes the tf between /map and /odom.

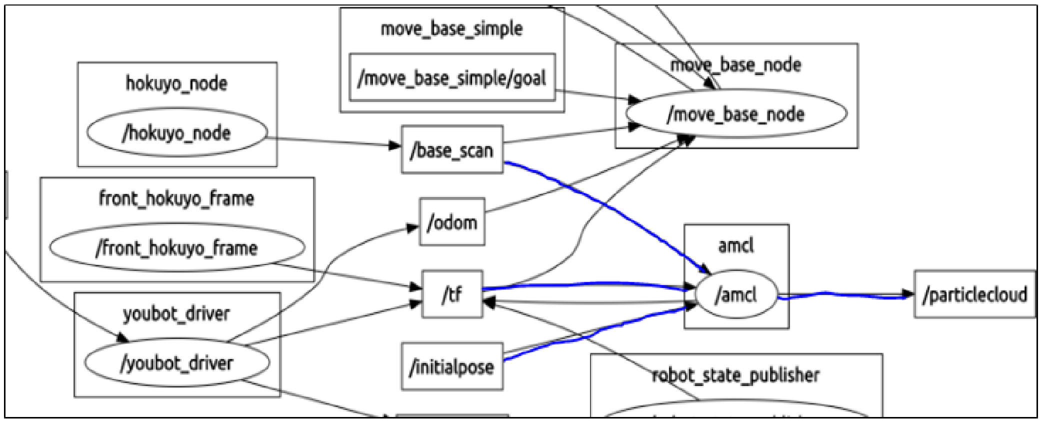
An **rqt_tf_tree** provides a GUI plugin for visualizing the ROS TF frame tree.


map
Broadcaster: /slam_gmapping
Average rate: 20.667
Buffer length: 1.5
Most recent transform: 1467021592.43
Oldest transform: 1467021590.93
odom
Broadcaster: /youbot_driver
Average rate: 28.035
Buffer length: 1.32
Most recent transform: 1467021592.18
Oldest transform: 1467021590.86
base_footprint
Broadcaster: /robot_state_publisher
Average rate: 50.668
Buffer length: 1.52
Most recent transform: 1467021592.89
Oldest transform: 1467021591.37
Broadcaster: /front_hokuyo_frame
Average rate: 10.639
Buffer length: 1.504
Most recent transform: 1467021592.48
Oldest transform: 1467021590.97
base_link
laser
Broadcaster: /robot_state_publisher
Average rate: 5.927
Buffer length: 0.844
Broadcaster: /robot_state_publisher
Average rate: 50.668
Buffer length: 1.52
Broadcaster: /robot_state_publisher
Average rate: 5.927
Buffer length: 0.844
Broadcaster: /robot_state_publish
Average rate: 5.927
Buffer length: 0.844
A portion of the tf tree for slam_gmapping

The TF tree shows the relations between various frames and the corresponding publishing rates. As all the publishing rates are reasonably close, the SLAM operation is successful.

6.4. Localization using an existing map – experimental results

Refer the lab manual for the instructions.

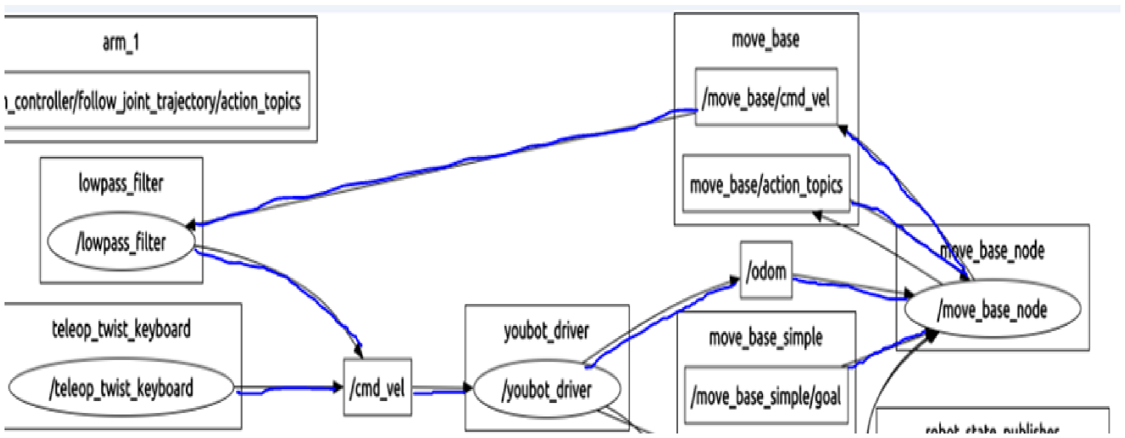


Rqt_graph of amcl

The **amcl** node takes in /base_scan, /tf and /intialpose as input and produces localization information under the topic /particlecloud. amcl also publishes the tf between /map and /odom. /intialpose contains the intial pose of the robot that we input using Rviz.

6.5. Autonomous navigation – experimental results

Refer the manual for instructions and further details.



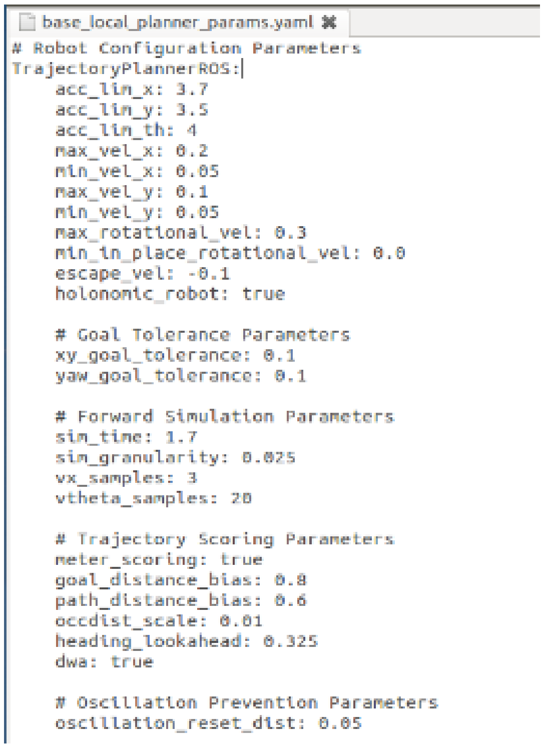
Rqt_graph of move_base

The **move_base** node takes in /odom, /move_base_simple/goal, /move_base/cmd_vel topics as input.
/odom contains the odometry data of the youbot, published by youbot_driver.

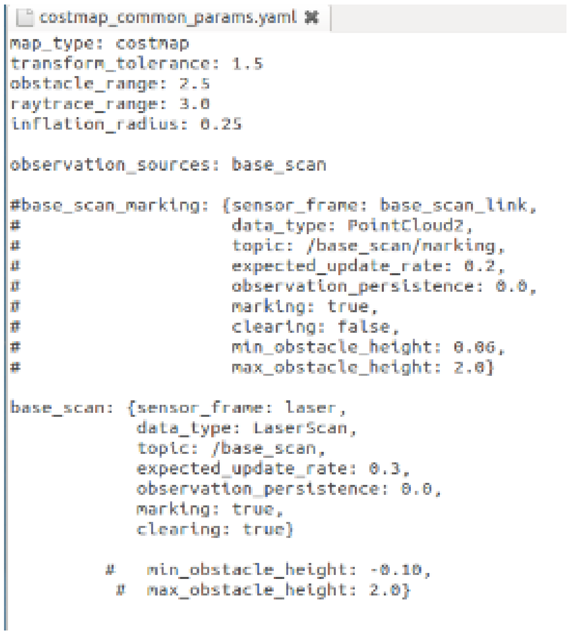
/move_base/cmd_vel contains the velocity command that the path planner publishes for the base. /move_base_simple/goal contains the goal pose in the map that we input in Rviz.

6.6. Optimizing the path planner

The **base_local_planner** parameters were modified to utilize the Omni-direction property of youBot. The acceleration limits were increased, and the Dynamic Window Approach was set to true to obtain accurate results at such high values.
Goal location accuracy was improved, and rotate-recovery mode on reaching the goal was ceased. The velocities of the bot for autonomous navigation was also lowered.
The common costmap parameters were modified, transform tolerance was increased to remove the warning due to delay in publishing transforms. The inflation radius was increased to keep the youBot away from the obstacles. The expected_update_rate parameter for each observation source was set based on the rate at which the sensor actually publishes.

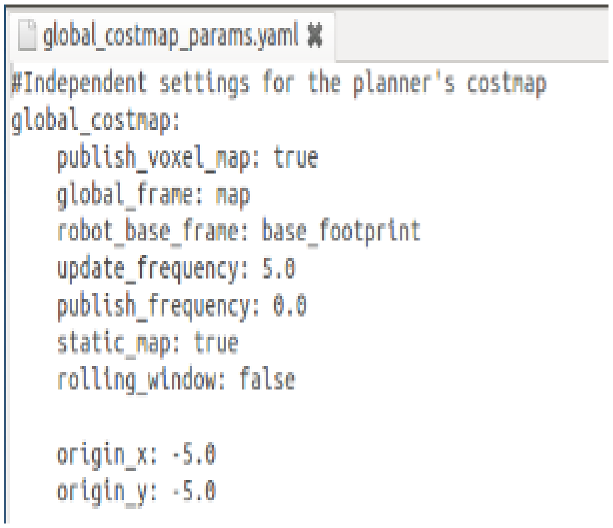

base_local_planner_params.yaml

```
# Robot Configuration Parameters
TrajectoryPlannerROS:
    acc_lim_x: 3.7
    acc_lim_y: 3.5
    acc_lim_th: 4
    max_vel_x: 0.2
    min_vel_x: 0.05
    max_vel_y: 0.1
    min_vel_y: 0.05
    max_rotational_vel: 0.3
    min_in_place_rotational_vel: 0.0
    escape_vel: -0.1
    holonomic_robot: true

    # Goal Tolerance Parameters
    xy_goal_tolerance: 0.1
    yaw_goal_tolerance: 0.1

    # Forward Simulation Parameters
    sim_time: 1.7
    sim_granularity: 0.025
    vx_samples: 3
    vtheta_samples: 20

    # Trajectory Scoring Parameters
    meter_scoring: true
    goal_distance_bias: 0.8
    path_distance_bias: 0.6
    occdist_scale: 0.01
    heading_lookahead: 0.325
    dwa: true

    # Oscillation Prevention Parameters
    oscillation_reset_dist: 0.05
```

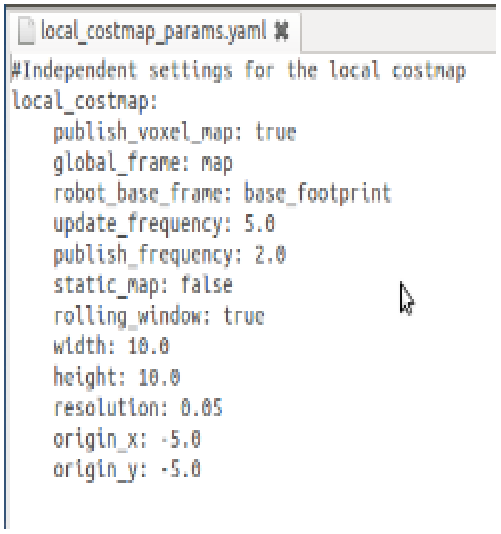

costmap_common_params.yaml

```
map_type: costmap
transform_tolerance: 1.5
obstacle_range: 2.5
raytrace_range: 3.0
inflation_radius: 0.25

observation_sources: base_scan

#base_scan_marking: {sensor_frame: base_scan_link,
#                    data_type: PointCloud2,
#                    topic: /base_scan/marking,
#                    expected_update_rate: 0.2,
#                    observation_persistence: 0.0,
#                    marking: true,
#                    clearing: false,
#                    min_obstacle_height: 0.06,
#                    max_obstacle_height: 2.0}

base_scan: {sensor_frame: laser,
            data_type: LaserScan,
            topic: /base_scan,
            expected_update_rate: 0.3,
            observation_persistence: 0.0,
            marking: true,
            clearing: true}

         #   min_obstacle_height: -0.10,
          #  max_obstacle_height: 2.0}
```

The update and publish frequencies were increased to keep the costmap up to date more accurately. The origin was changed from the default (0,0) to the negative XY quadrant to include all the sensor values. Setting the static map parameter to true means that the node will take an outside map source for navigation. That map could from SLAM or map_server.



```
#Independent settings for the planner's costmap
global_costmap:
    publish_voxel_map: true
    global_frame: map
    robot_base_frame: base_footprint
    update_frequency: 5.0
    publish_frequency: 0.0
    static_map: true
    rolling_window: false

    origin_x: -5.0
    origin_y: -5.0
```



```
#Independent settings for the local costmap
local_costmap:
    publish_voxel_map: true
    global_frame: map
    robot_base_frame: base_footprint
    update_frequency: 5.0
    publish_frequency: 2.0
    static_map: false
    rolling_window: true
    width: 10.0
    height: 10.0
    resolution: 0.05
    origin_x: -5.0
    origin_y: -5.0
```

6.7. Challenges faced

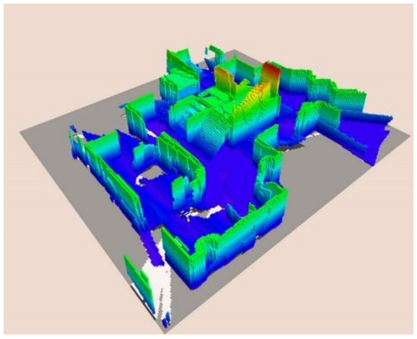
1. Attempt to implement a new local path planner was not successful. The **base_local_planner** was replaced by **eband_local_planner**, which implements the Elastic Band method on the SE2 manifold. This planner was able to achieve obstacle avoidance, but the youBot failed to stop at the goal location.

2. Attempt to use a new global path planner based on a **relaxed A* algorithm** was not successful. The path planner was added as a Plugin in ROS, under the move_base node. But the move_base node could not be modified to use it as the global path planner. The trajectory path could not be visualized in Rviz.

3. The robot always showed tendency to perform autonomous navigation like a differential drive robot rather than fully utilizing the Omni-directional property.

7. 3D Mapping using a rotating LIDAR

7.1. Introduction



Existing work on 3D Mapping has been done using a combination of Kinect sensor and IMU on a mobile base. The attempt was to generate a 3D map using a LIDAR-Servo combination with a youbot. Attaching a dynamixel servo to the sensor would provide the required pitching motion to the LIDAR to capture the 3D range data and generate the point cloud.

7.2. ROS package

It was found that octomap_server node was the ideal package to perform 3D mapping in ROS. ROS node - **Octomap_mapping** - allows to incrementally build 3D OctoMaps, and provides map saving in the node octomap_saver. The map implementation is based on an octree. The map is able to model arbitrary environments without prior assumptions about it. The representation models occupied areas as well as free space.
Octomap_server requires **PointCloud2** data as input. So the laser scan data was converted to PointCloud2 data using **laser_assembler** package.

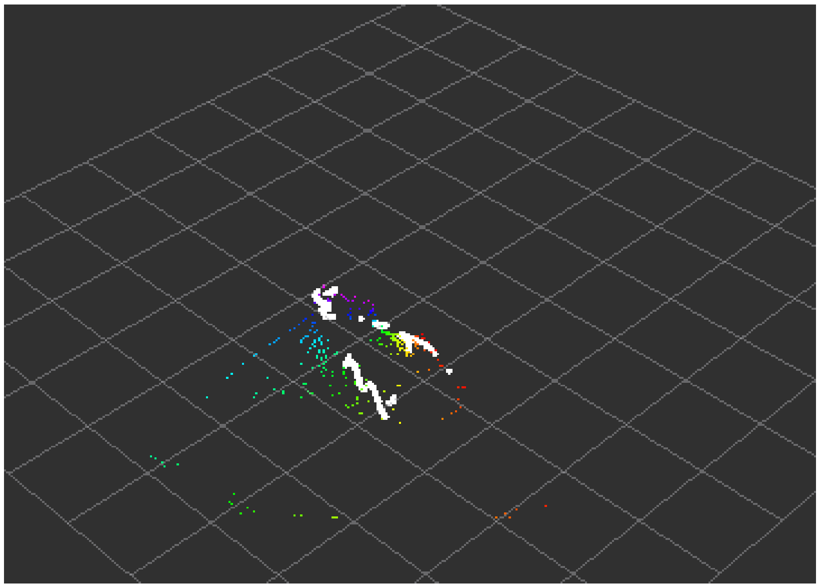
A detailed description about this algorithm is mentioned in [10].

7.3. Laser_assembler package

The **laser_assembler** package provides nodes that listen to streams of scans and then assemble them into a larger 3D Cartesian coordinate (XYZ) point cloud.

To publish the assembled cloud data, a ROS service must be called

The **periodic_snapshotter**.**cpp** example node was written to convert laser_scan to PointCloud data by calling the assemble_scans service. But octomap_server requires PointCloud2 data, so example node was modified to call assemble_scans2 service and produce PointCloud2 output.


Laser scan data (white) and assembled cloud data (rainbow) as seen in Rviz

7.4. Challenges faced

1. The **periodic_snapshotter** node always showed a delay in publishing assembled_cloud data from laser scan data
2. Transform rate showed erroneous values when the laser sensor was launched along with the dynamic TF publisher.
3. The dynamic TF created for the servo does not show any sync with the servo motion
4. **Octomap_mapping** generated no map due to error in TF
5. Combining the youBot base motion with the octomap node was not possible.

8. Conclusion and Future scope

1. The sensor and actuator were successfully interfaced with ROS. A ROS wrapper for running the youBot was also installed.

2. The KUKA youBot was capable of achieving 2D SLAM and navigation using a LIDAR sensor. Occupancy grid maps of fairly good accuracy was obtained. The robot was also able to self localize in an unknown environment by providing an accurate map and intial pose. The robot also achieved autonomous navigation to the desired goal location at a reasonable pace. The path planner enabled the robot to handle both static and dynamic obstacles in its path.

3. For 3D mapping, the required conversion from laserscan to PointCloud2 was achieved, but not in real time.

4. All the required information regarding youBot SLAM and navigation can be obtained by publishing the corresponding ROS topics.

Some of the future work to be done:

1. Implementing the new global path planner using the relaxed A* algorithm on the youBot.
2. Stretching the PointCloud2 data to 3D space and visualizing it.
3. Creating a dynamic TF publisher that causes the servo frame to mimic the actual motion of the servo.
4. Generating a 3D map by feeding the stretched PointCloud2 data to the Octomap node.

9. References

1. wiki.ros.org
2. www.youbot-store.com/wiki
3. Learning ROS for Robotics Programming, Book by Aaron Martinez and Enrique Fernandez
4. ROS By Example HYDRO – Volume 1 Book by R.Patrick Goebel
5. Probabilistic Robotics, Book by Dieter Fox, Sebastian Thrun, and Wolfram Burgard
6. Introduction to Autonomous Mobile Robots, Book by Roland Siegwart
7. G. Grisetti, C. Stachniss and W. Burgard, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters," in IEEE Transactions on Robotics, vol. 23, no. 1, pp. 34-46, Feb. 2007.
8. J. M. Santos, D. Portugal and R. P. Rocha, "An evaluation of 2D SLAM techniques available in Robot Operating System," 2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Linkoping, 2013, pp. 1-6.
9. S. Kohlbrecher, O. von Stryk, J. Meyer and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, Kyoto, 2011, pp. 155-160.
10. Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. 2013. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. Auton. Robots 34, 3 (April 2013), 189-206