INTERNSHIP REPORT

(May 03 - July,22 2019)

RAJENDRA SINGH
COMPUTER SCIENCE AND ENGINEERING
FOURTH YEAR, IIT PALAKKAD
111601017

PREFACE

The present report is the outcome of the Internship Program at Infinity Labs, UST Global, trivandrum, Kerala. The objective of the internship program was to familiarize the student with the implementation of the knowledge she/he earned on the campus. Report describes in detail what we learnt and implemented during internship period of three months.

ACKNOWLEDGEMENT

I would like to thank all the people who helped me during the internship, especially my project **mentor Ashok Nair** for the time he devoted supervising and guiding me in my project. I would like to thank all the lab members for contributing to a very positive spirit of friendship and noncompetition. I would like to thank UST global for giving me the opportunity to work on the topic of my utmost interest.
I would like to thank the developers of ROS, Gazebo, Gmapping, RTABMap, moveit and Rviz for developing such powerful tools.

ABOUT RESEARCH LAB

UST Global® is a leading digital technology company that provides advanced computing and digital services to large private and public enterprises around the world. Driven by a larger purpose of Transforming Lives and the philosophy of “fewer clients, more attention”, it brings in the entrepreneurial spirit that seeks the fastest path to value in today’s digital economy. Their innovative technology services and pioneering social programs make them unique. UST Global is headquartered in Aliso Viejo, California and operates in 21 countries. Their clients include Fortune 500 companies in Banking and Financial Services, Healthcare, Insurance, Retail, High Technology, Manufacturing, Shipping, and Telecom.
The Infinity LabsTM are a core ingredient of UST Global’s Innovation-as-a-Service offering. It creates a confluence of Digital Technologies, Human Centered Design, Artificial Intelligence, the company’s ecosystem of partners and cutting-edge R&D from academia. The Infinity LabsTM provides a platform for UST Global employees to get exposed to cutting-edge technologies, gain access to world-class R&D institutions and disruptive startups from around the world. The labs also have an experience center where clients can get an immersive experience of UST Global’s capabilities and services.

ABSTRACT

During my internship period of three months at UST Global I worked in the area of robotics, this includes slam algorithms, robotic manipulators and swarm robotics.
As autonomous driving is rapidly becoming the next major challenge in the automotive industry, the problem of Simultaneous Localization And Mapping (SLAM) has never been more relevant than it is today. This report presents the idea of examining SLAM algorithms by implementing such an algorithm on a radio-controlled car which has been fitted with sensors and microcontrollers. The software architecture of this small-scale vehicle is based on the Robot Operating System (ROS), an open-source framework designed to be used in robotic applications. This report also presents the idea of combining various sensor data to get a more accurate and reliable result for slam algorithms.
In the end it describes the idea of using robotic manipulator for vision-based pick and place, where we use forward and inverse kinematics to plan the movement of uarm swift pro robotic arm.

TABLE OF CONTENTS

➢ Certificate

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviations		Description
ROS	-	Robot Operating System
LIDAR	-	Light detection and ranging
SLAM	-	Simultaneous localization and mapping
IMU	-	Inertial measurement unit
PDF	-	Probability Density Function
EKF	-	Extended Kalman Filter
GPS	-	Global Positioning System
Rviz	-	3D visualization tool for ROS
DOF	-	Degrees of Freedom
PCL	-	Point Cloud Library (C++ library)
RGB-D	-	Red Green Blue Depth
Voxel	-	Cubic subdivision of volume
Odom	-	Odometry
amcl	-	Adaptive Monte Carlo localization
RTABMAP	-	Real-Time Appearance-Based Mapping
ICP	-	iterative closest point
CAN	-	continuous attractor network
Pkg(ros)	-	Package
Msg(ros)	-	messages
V-REP	-	Virtual Robot Experimentation Platform
IP	-	Internet Protocol)
DNS	-	Domain Name System services
DHCP	-	Dynamic Host Configuration Protocol

1. Robot Operating System

1.1 Introduction

Robot Operating System(ROS) is a meta-operating system for a robot. It provides services that one would expect from an operating system, including hardware abstraction, device drivers, libraries, visualizers, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. It is named as a meta-operating system because it is something between an operating system and middleware. It provides not only standard operating system services (like hardware abstraction) but also high-level functionalities like asynchronous and synchronous calls, a centralized database, a robot configuration system, etc. ROS can be interpreted also as a software framework for robot software development, providing the operating system.ROS is based on a Unix-like philosophy of building many small tools that are designed to work together. ROS grows out of a collaboration between industry and academia and is a novel blend of professional software development practices and the latest research results.

1.2 Why ROS?

Because creating a truly robust, general-purpose robot software is hard. From the robot's perspective, problems that seem trivial to humans often vary wildly between instances of tasks and environments. Dealing with these variations is so hard that no single individual, laboratory, or institution can hope to do it on their own.

As a result, ROS was built from the ground up to encourage collaborative robotics software development. For example, one laboratory might have experts in mapping indoor environments and could contribute a world-class system for producing maps. Another group might have experts at using maps to navigate, and yet another group might have discovered a computer vision approach that works well for recognizing small objects in clutter. ROS was designed specifically for groups like these to collaborate and build upon each other's work, as is described throughout this site.

1.3 Programming with ROS

ROS is language-independent. The libraries are wrapped with a thin message-passing layer that enables them to be used by and make use of other ROS nodes. Messages are passed peer to peer and are not based on a specific programming language; nodes can be

written in C++, Python, C, LISP, Octave, Lua or any other language for which someone has written a ROS wrapper.
ROS software is organized into packages. Before writing any programs, the first step is to create a workspace that holds the packages, and then create the package itself. An example of creating a package is catkin_create_pkg example1, which will create a package named example1. Created packages should live in a workspace directory.

1.4 Basic ROS concepts and terminology

- Nodes : A node really isn't much more than an executable file within a ROS package. ROS nodes use a ROS client library to communicate with other nodes. Nodes can publish or subscribe to a Topic. Nodes can also provide or use a Service. (ros_graph)
- Message : ROS data type used when subscribing or publishing to a topic. E.g LaserScan message look like:
    - std_msgs/Header header
    - float32 angle_min
    - float32 angle_max
    - float32 angle_increment
    - float32 time_increment
    - float32 scan_time
    - float32 range_min
    - float32 range_max
    - float32[ ] ranges
    - float32[ ] intensities
- Topics : Nodes can publish messages to a topic as well as subscribe to a topic to receive messages.
- Publisher : Node which is publishing on the topic.
- Subscriber : Node which is subscribing to the topic.

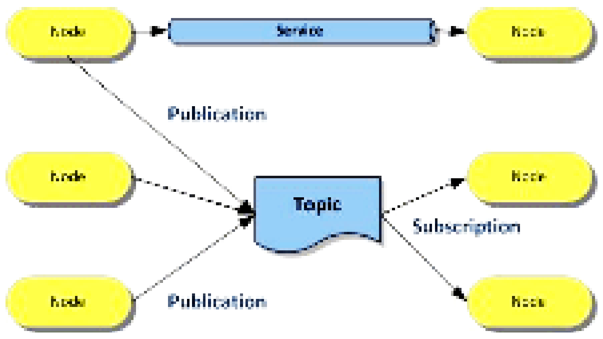


Figure 1. Relation between nodes, topics and services.

- Master: Name service for ROS (i.e. helps nodes find each other)
- Services : Like ros topic but will receive msg only when asked for.
- Catkin_create_pkg : command to create ros pkg

- Client Libraries : ROS client libraries allow nodes written in different programming languages to communicate:
    - rospy = python client library
    - roscpp = c++ client library

- Roscore : roscore is the first thing you should run when using ROS.
- URDF : file in which we describe robots links and joints
- Gazebo and Vrep: software for physics simulations which is supported by ros.
- Rviz : visualisation software for ros, where we can visualise robot, mapping, planning, navigation and more things.

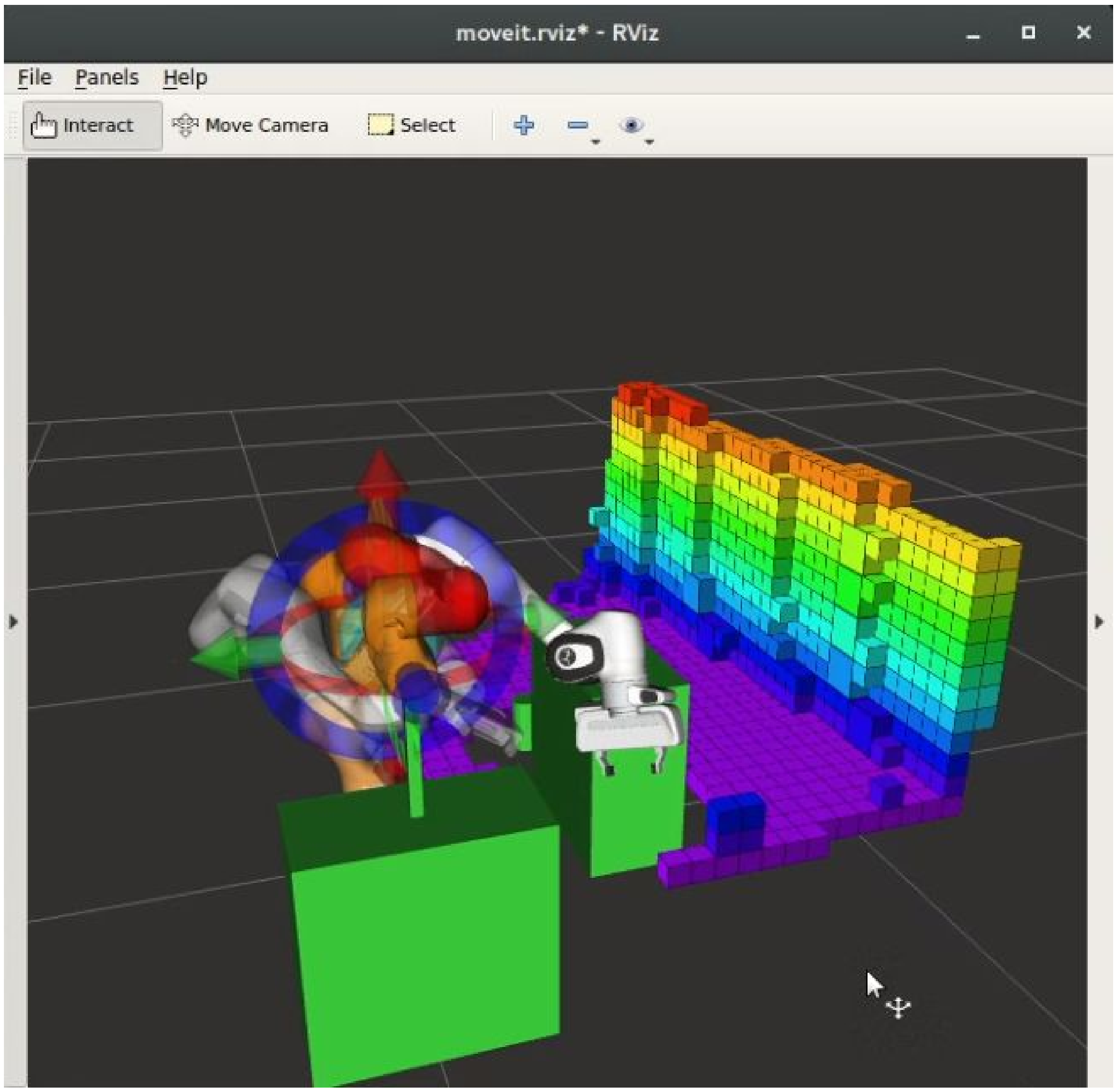


Figure 2. Example of visualisation in Rviz

- TF : tf is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.

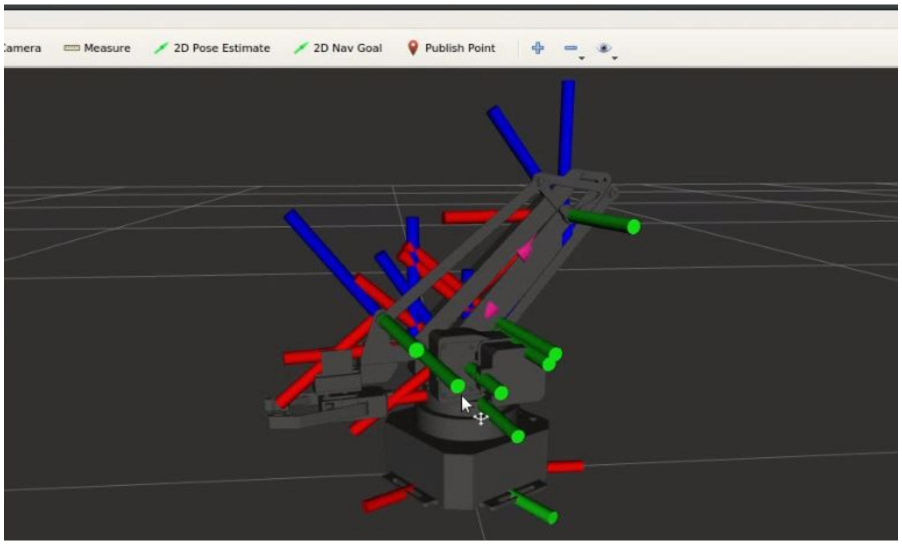


Figure 3. Example of frame transformation

- ros_control : The ros_control packages takes as input the joint state data from your robot's actuator's encoders and an input set point. It uses a generic control loop feedback mechanism, typically a PID controller, to control the output, typically effort, sent to your actuators. ros_control gets more complicated for physical mechanisms that do not have one-to-one mapping of joint positions, efforts, etc but these scenarios are accounted for using transmissions.

2. ROS based surveillance robot : RAWBOT

demo

Rawbot is a ros based surveillance robot, I started back in December 2018. Aim of this project is to develop autonomous ROS based surveilling all terrain robot. Features in RAWBOT includes:

I. Teleoperated robot
II. Face recognition
III. Audio command, feedback and communication
   A. Google assistant
   B. Audio topic
   C. Espeak
IV. Multiple camera streams
V. Distributed computation
   A. Raspberry pi
   B. Arduino
VI. Limitation
   A. Not yet autonomous
   B. No robotic manipulator on board.

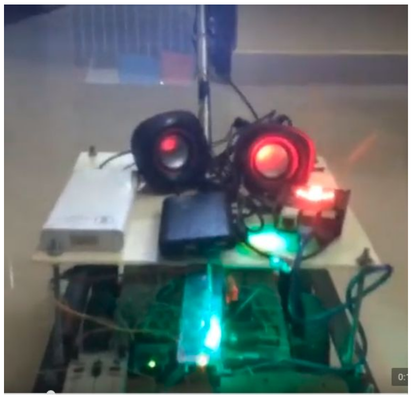


Figure 4. Rawbot

VII. Conclusion : To remove above limitation, I worked on SLAM and Robotic manipulator during my internship period, which will be discussed in detail below.

3. SLAM

3.1 Theory

3.1.1 Motivation

Mobile robots are gaining significant relevance and starting to enter our daily life. For example, Vacuum cleaner,robotic lawnmower. Besides, they become more popular in the industry. Many companies started to use automated guided vehicles or mobile manipulators in their warehouses. Moreover, mobile robots are used in explorations of caves, pyramids, reefs or similar things. Another possible application of mobile robots in military operations where they could be used for transportation, reconnaissance or even fighting. The main request to a mobile robot obviously is mobility. The robot has to be capable of moving from its current pose, defined by its position and orientation, to a desired target configuration. In order to achieve this, it needs to find a valid collision-free path connecting the corresponding configurations. Moreover, there may be other demands to the path like to be as short, as fast or as safe as possible. If the robot wants to find such a path it needs access to several information. Firstly it needs information about itself such as its own size and its manoeuvrability. Furthermore, the environment including the current pose and the target pose have to be known. For this purpose mobile robots usually use a map. Now given this information path can be found. These paths can be calculated with several path planning algorithms. Planning algorithms usually work in discrete domains, which is why often trajectory optimization methods are used in the second step in order to smoother these paths. SLAM methods can create a map of an unknown environment and localize the robot in this map at once. That means they allow the robot to deal with any unknown environment. Therefore they are very popular and often used in mobile robotics. With SLAM there is no more need to create a map and hand it to the robot in order to navigate it since the robot develops its own.

3.1.2 Introduction

Human beings get the information about their surrounding through vision, hearing the voice through ears, smelling through the nose and they do feel the strength of objects through touch. In general human beings get information about reality through senses. A robot cannot explore an unknown environment unless it is provided with some sensing sources to get information about the environment. There are different kinds of sensors used to make a robot capable of sensing a wide range of environments: Odometers, Laser range finders, Global Position System (GPS), Inertial Measurement Units (IMU), Sound Navigation and Ranging (Sonar) and cameras. The map of the environment is a basic need for a robot to perform actions like moving room to room, picking an object from one place and taking it to another one. To perform such actions, the robot should not only know about the environment but while it is moving it should also be aware of its own location in that environment. The aim to achieve is the navigation of the robot in a new and unknown environment by using the ROS. Robot builds the map, localizes itself on the map and performs navigation.

Here we describe the basic features of a generic SLAM algorithm, as well as introducing a few SLAM algorithms. Simultaneous localization and mapping is a problem where a moving object needs to build a map of an unknown environment, while simultaneously calculating its position within this map. There are several areas which could benefit from having autonomous vehicles with SLAM algorithms implemented. Examples would be the mining industry, underwater exploration, and planetary exploration.

3.1.3 Formulation

The SLAM problem, in general, can be formulated using a probability density function denoted

$$p(xt, m|z1: t, u1:t)$$

where,

xt	- position of the vehicle at time t
m	- map
z1:t	- vector of all measurements(Observations)
u1:t	- vector of the control signals of the vehicle(control commands or odometry)

3.1.4 Data association

Basically, the concept data association is to investigate the relationship between older data and new data gathered. In a SLAM context, it is of necessity to relate older measurements to newer measurements. This enables the process of determining the locations of landmarks in the environment, and thus this also gives information regarding the robot's position within the map.

Simultaneous localization and mapping.

3.1.5 Loop closure

The concept of loop closure in a SLAM context is the ability of a vehicle to recognize that a location has already been visited. By applying a loop closure algorithm, the accuracy of both the map and the vehicle's position within the map can be increased. However, this is not an easy task to perform, due to the fact that the operating environment of the vehicle could contain similar structural circumstances as previously visited locations. In that case, if the loop closure algorithm performs poorly, it could lead to a faulty loop closure, which could corrupt both the map as well as the pose of the vehicle within the map.

3.1.6 Full SLAM and online SLAM

There are two different types of SLAM problems. The online SLAM problem of which only the current pose xt(x at time t) and the map m are expressed, given the control input u1:t and measurements z1: t. As well as the full SLAM problem which expresses the entire trajectory of the robot. The PDF of the full SLAM problem is denoted as p(x0: t, m|z1:t , u1:t), where all the poses of the robot are considered, including its initial pose x0.

3.1.7 General models for SLAM problem

Challenge of SLAM is to compute the robots state xt and the map m depending on previous observations z0:t and control commands u0:t−1. For this to happen a motion model returning the current state xt depending on the previous state xt−1 and the last control command u_{t-1} is needed. The motion model is typically derived from the odometry data of the robot. Since odometry is sensitive to errors like non-round contours of the wheels, belt slip or inaccurate calibration this model is not deterministic. However, it can describe the belief of the state $p(x_t)$ as conditional probability

$$p(x_t) := P(x_t | x_{t-1}, u_{t-1})$$

Furthermore an observation model returning the current expected sensor output z_t depending on the state x_t and the map m has to be known. Due to inaccuracies of the sensors, this model is also not deterministic, but can be described as conditional probability

$$p(z_t) := P(z_t | x_t, m)$$

These models being non-deterministic means that everything we calculate using them will not be absolute. That implies that the SLAM problem can not be solved absolutely. It is not possible to compute the actual state or map. But what can be estimated is a belief over state and map represented by a probability. Assuming we know the motion and transition model of a certain robot and all previous states $x_{0:t}$, the belief over the map m could be easily computed as

$$p(m) := P(m | x_{0:t}, z_{0:t}, u_{0:t-1})$$

Vice versa, if the map m and the models are known, the state x_t can be estimated as

$$p(x_t) := P(x_t | m, z_{0:t}, u_{0:t-1})$$

But in the SLAM problem the robot knows neither its location nor the surrounding environment. And it is not just that both are unknown, in fact they depend on each other, which makes SLAM a **chicken-egg-problem**. The challenge is to compute the belief over the map m and the state x_t simultaneously since they can not be estimated one after each other. Moreover, this has to be done carefully because wrong data association can lead to divergence. In fact there exist two different definitions of the SLAM problem. The first only seeks to recover the current pose x_t and is formulated as

$$\mathbf{p(x_t, m) := P(x_t, m | z_{0:t}, u_{0:t-1})}$$

This description is known as online SLAM. The other definition, which is called full SLAM, estimates the whole state trajectory $x_{0:t}$ as

$$\mathbf{p(x_{0:t}, m) := P(x_{0:t}, m | z_{0:t}, u_{0:t-1})}$$

Above two equations may seem impossible to solve at first. But since the solution of the SLAM problem is very significant for mobile robotics, it is a well-studied problem and there were several methods developed to solve it.

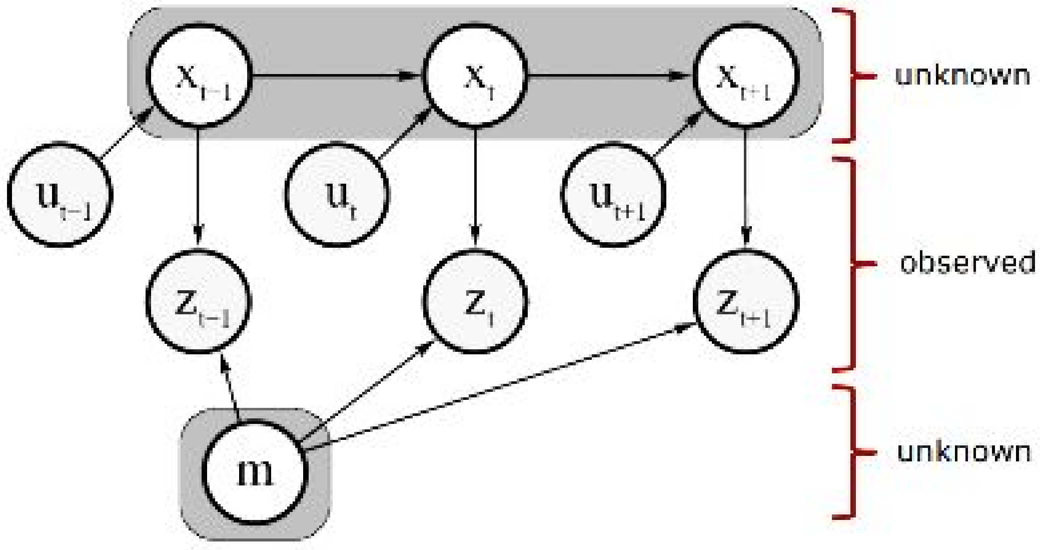
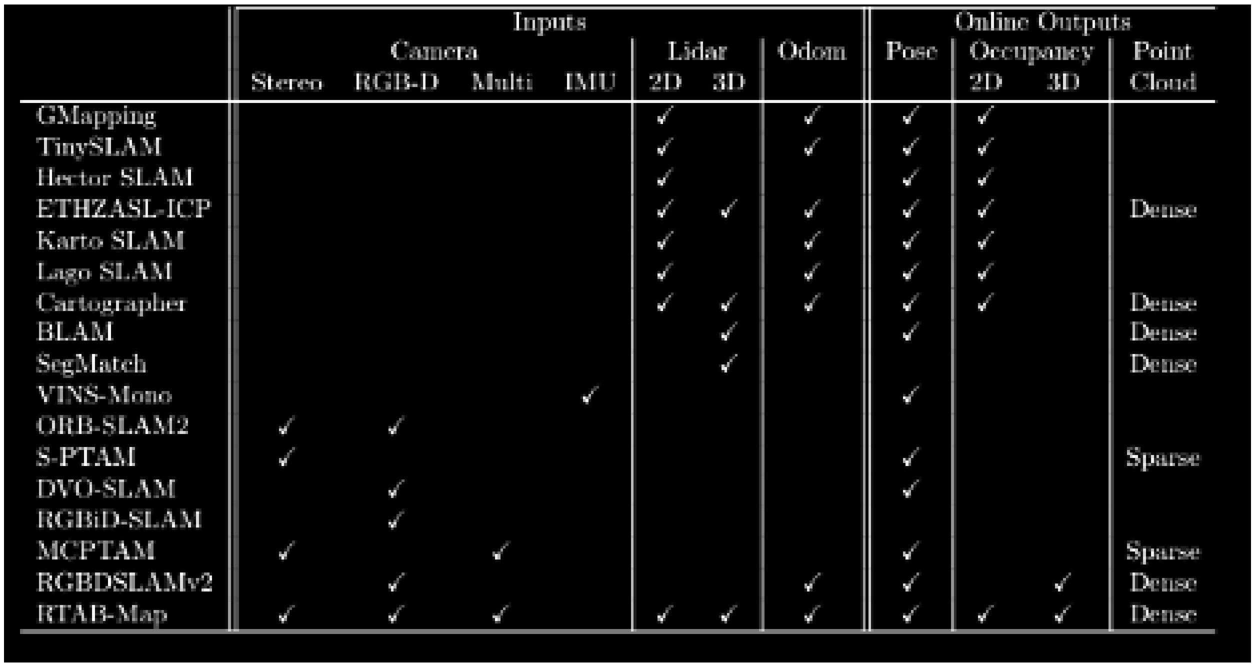


Figure 5. Relation between pose, observation, command and map.

A motion model for the SLAM problem can be described by xt = g(ut , xt−1) + δt where the current pose xt depends on the possible nonlinear function g(ut, xt−1), with ut being the control input, xt−1 the previous pose and δt being a random Gaussian variable with zero mean.
A measurement model for the SLAM problem can be described by: zt = h(xt, m) + δt where the current measurement zt, depends on the possible nonlinear function h(xt, m), where xt denotes the current pose, m represents the map, random Gaussian variable with zero mean.

3.1.8 Popular SLAM Approaches Available on ROS



	Inputs							Online Outputs			
	Camera				Lidar		Odom	Pose	Occupancy		Point Cloud
	Stereo	RGB-D	Multi	IMU	2D	3D			2D	3D	
GMapping					✓		✓	✓	✓		
TinySLAM					✓		✓	✓	✓		
Hector SLAM					✓			✓	✓		
ETHZASL-ICP					✓	✓	✓	✓	✓		Dense
Karto SLAM					✓		✓	✓	✓		
Lago SLAM					✓		✓	✓	✓		
Cartographer					✓	✓	✓	✓	✓		Dense
BLAM						✓		✓			Dense
SegMatch						✓					Dense
VINS-Mono				✓				✓			
ORB-SLAM2	✓	✓									
S-PTAM	✓							✓			Sparse
DVO-SLAM		✓						✓			
RGBiD-SLAM		✓									
MCPTAM	✓		✓					✓			Sparse
RGBDSLAMv2		✓					✓	✓		✓	Dense
RTAB-Map	✓	✓	✓		✓	✓	✓	✓	✓	✓	Dense

Figure 6. Comparison of various slam algorithms

There are a great variety of open-source SLAM approaches available through ROS.

• **GMapping [2007]** use a particle filter to estimate the robot trajectory. As long as there are enough estimated particles and the real position error corresponds to the covariance of the input odometry, the particle filter converges to a solution which represents well the environment, particularly for GMapping when there are loop closures. GMapping, being ROS's default SLAM approach, has been widely used to derive a 2D occupancy grid map of the environment from 2D laser scans. Once the map is created, it can be used with Adaptive Monte Carlo Localization for localization and autonomous navigation.
• **Hector SLAM [2011]** can create fast 2D occupancy grid maps from a 2D lidar with low computation resources. It has proven to generate very low-drift localization while mapping in real-world autonomous navigation scenarios. It can also use external sensors like an IMU to estimate the robot position in 3D. However, Hector SLAM is not exactly a full SLAM approach as it does not detect loop closures, and thus the map cannot be corrected when visiting back a previous localization. Hector SLAM does not need external odometry, which can be an advantage when the robot does not have one, but can be a disadvantage when operating in an environment without a lot of geometry constraints, limiting laser scan matching performance.
• **Karto SLAM [2010] and Google Cartographer [2016]** are lidar graph-based SLAM approaches. They can generate 2D occupancy grid from their graph representation. Google Cartographer can be also used as backpack mapping platform as it supports 3D lidars, thus providing a 3D point cloud output. While mapping, they create sub-maps that are linked by constraints in the graph. When a loop closure is detected, the position of the sub-maps are re-optimized to correct errors introduced by noise of the sensor and scan matching accuracy. Unlike Hector SLAM, external odometry can be provided to get more robust scan matching in environments with low geometry complexity.

• **LSD-SLAM** is a novel, direct monocular SLAM technique: Instead of using keypoints, it directly operates on image intensities both for tracking and mapping. The camera is tracked using direct image alignment, while geometry is estimated in the form of semi-dense depth maps, obtained by filtering over many pixel wise stereo comparisons.

3.2 Slam implementation:

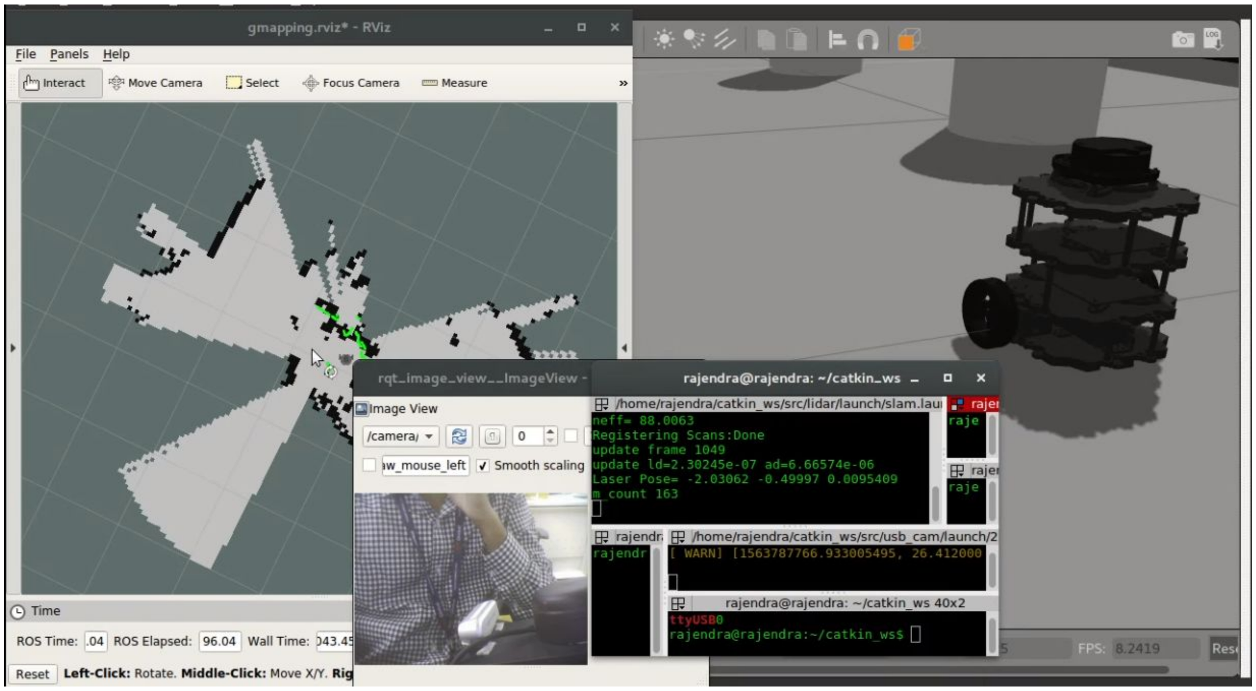
3.2.1 Gmapping slam using lidar

I used RPLidar A208 model to create map the environment. I implemented gmapping slam to create the map and tested the navigation on turtlebot burger model in gazebo simulation. Detailed description of which is as below:

a. Firstly I create ros node to read the lidar data and publish scan message on /scan topic.
b. Here we use the turtlebot burger model in gazebo, to get the relevant tf of robot.
c. Instead of subscribing to the scan topic generated by gazebo model we subscribe our real robot lidar sensor.
d. Now let create launch file for mapping where we do the following: demo
    i. Launch the turtlebot world and spawn the burger model.
    ii. Now we run the instance of the robot state publisher for getting relevant transforms.

iii. Load the urdf to robot description param.
iv. Launch lidar node.
v. Now we run the slam_gmapping node from gmapping package with appropriate base, odom and map frame.
vi. Launch the rviz node.
vii. Now we move the robot around using the teleop node, create the map.
viii. Once we’re happy with the generated map, we save it using the map_saver node from map_server pkg.

e. Now once we have the good map of environment, we navigate around the space: demo
    i. Here also we run the similar launch file as in mapping, just that we do not launch slam_gmapping node.
    ii. We launch the map_server node for publishing the already saved map, as global map.
    iii. We also run the instance of the acml node which will localise the robot in the surrounding environment, publish robot pose as covariance vector, which will improve over time.
    iv. Then we run the move_base node which is responsible for robot path planning. Here we used 2 types of DWAPlanner:
        1. Global planner : It is responsible for giving the global optimised path between start and destination pose.
        2. Local planner: It is responsible for planning the path locally for instance, when there also people moving in the same environment. In this case we can not navigate based on the global plan is based out one time static map.
    v. Here our case we rviz gui tool to initial state estimation, and giving the goal.


gmapping.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure
Time
ROS Time: .04 ROS Elapsed: 96.04 Wall Time: 043.45
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Rig
rqt_image_view__ImageView
Image View
/camera/
0
w_mouse_left Smooth scaling
rajendra@rajendra: ~/catkin_ws
FPS: 8.2419
Res

```
/home/rajendra/catkin_ws/src/lidar/launch/slam.lau
neff= 88.0063
Registering Scans:Done
update frame 1049
update ld=2.30245e-07 ad=6.66574e-06
Laser Pose= -2.03062 -0.49997 0.0095409
m_count 163

/home/rajendra/catkin_ws/src/usb_cam/launch/2
[ WARN] [1563787766.933005495, 26.412000

rajendra@rajendra: ~/catkin_ws 40x2
ttyUSB0
rajendra@rajendra:~/catkin_ws$
```

Figure 7. Mapping with lidar

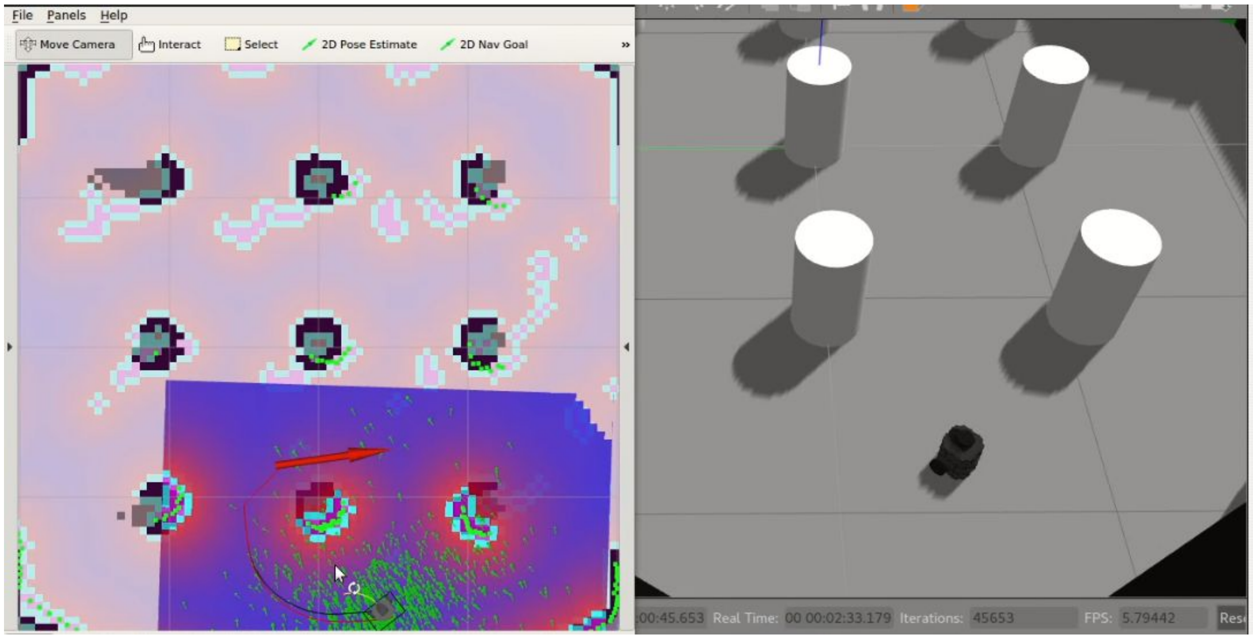

File Panels Help
Move Camera Interact Select 2D Pose Estimate 2D Nav Goal
00:45.653 Real Time: 00 00:02:33.179 Iterations: 45653 FPS: 5.79442 Res

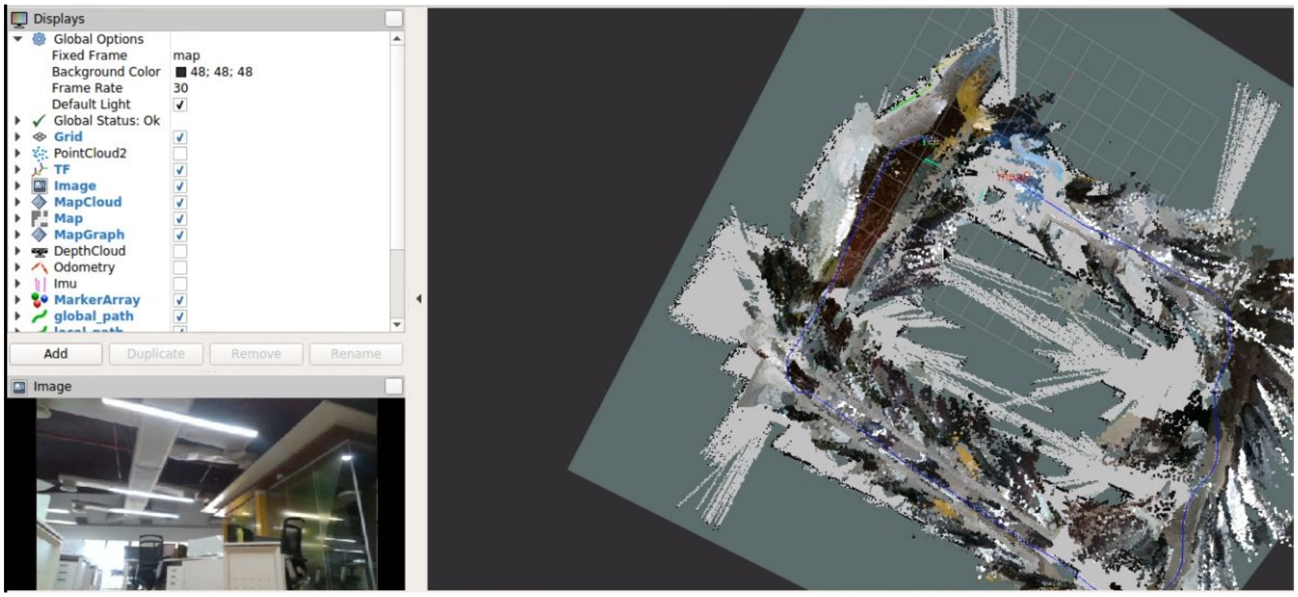
Figure 8. Navigation in gazebo environment

3.2.2 RTAB-MAP slam using 3D Camera [demo]

I used intel realsense D435 to get the pointcloud data of real environment. This data was used to create the 3D map to environment using RTABMAP slam algorithm. Real-Time Appearance-Based Mapping(RTAB-Map) is a RGB-D, Stereo and Lidar Graph-Based SLAM approach based on an incremental appearance-based loop closure detector. The loop closure detector uses a bag-of-words approach to determinate how likely a new image comes from a previous location or a new location. When a loop closure hypothesis is accepted, a new constraint is added to the map's graph, then a graph optimizer minimizes the errors in the map. A memory management approach is used to limit the number of locations used for loop closure detection and graph optimization, so that real-time constraints on large-scale environnements are always respected. RTAB-Map can be used alone with a handheld Kinect, a stereo camera or a 3D lidar for 6DoF mapping, or on a robot equipped with a laser rangefinder for 3DoF mapping. I followed these steps for creating 3d map of the environment:

a. First of we launch node which read the rgb and depth image from the 3d camera and publish as rostopic.
b. We run rgbd_sync node to sync these images if they aren't already.
c. We can run the rtabmap is these two modes:
    i. Visual odom : This is feature based odometry, where we use the image feature to localise the robot in the environment. An RGBD odometry finds the camera movement between two consecutive RGBD image pairs. The input are two instances of RGBDImage. The output is the motion in the form of a rigid body transformation.
    ii. Icp_odom : Its use iterative closest point(icp) algorithm to get the odom data.
d. We use the lidar data along with pointcloud2 data to get more accurate mapping of the environment.
e. Then run the rtabmap node with appropriate sensor topics, frames and parameter.
f. For visualisation of the mapping we can either use rtabmapviz or rviz.
g. Then we move the robot around the work space to create the map. This can be done via teleop node or manually. Map will auto correct itself when it detects the loop closure. Many times this will result in odom frame drifting form map frame in discrete move.
h. Once we are happy with the obtained map, we stop the mapping. This will save the 3d map,poses and camera images .db file which can be used later during the navigation. demo
i. We can also further do post processing on the obtained to map to further refine it. demo

Similar to the mapping we can also run the rtabmap in localisation mode where its will use previously generated map to localise in environment. Also we can give the goal pose directly through rviz path planning.



```
Global Options
  Fixed Frame        map
  Background Color   ■ 48; 48; 48
  Frame Rate         30
  Default Light      ✓
Global Status: Ok
Grid                 ✓
PointCloud2
TF                   ✓
Image                ✓
MapCloud             ✓
Map                  ✓
MapGraph             ✓
DepthCloud
Odometry
Imu
MarkerArray          ✓
global_path          ✓
```

Figure 9. Mapping using realsense D435 and rplidar A208

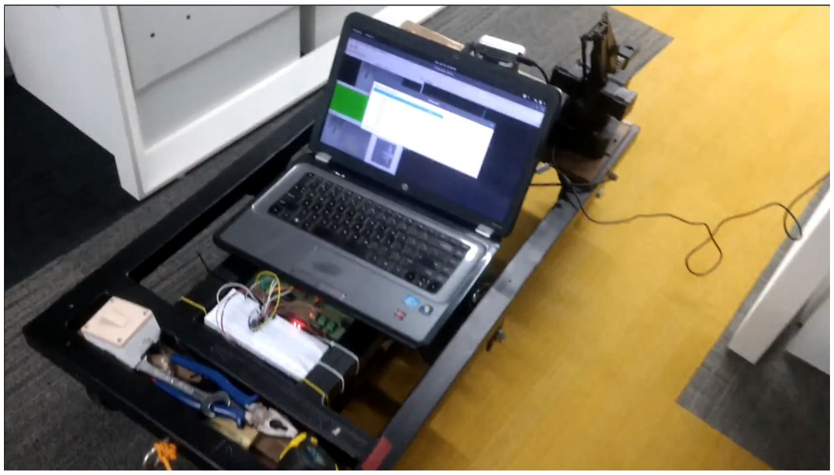
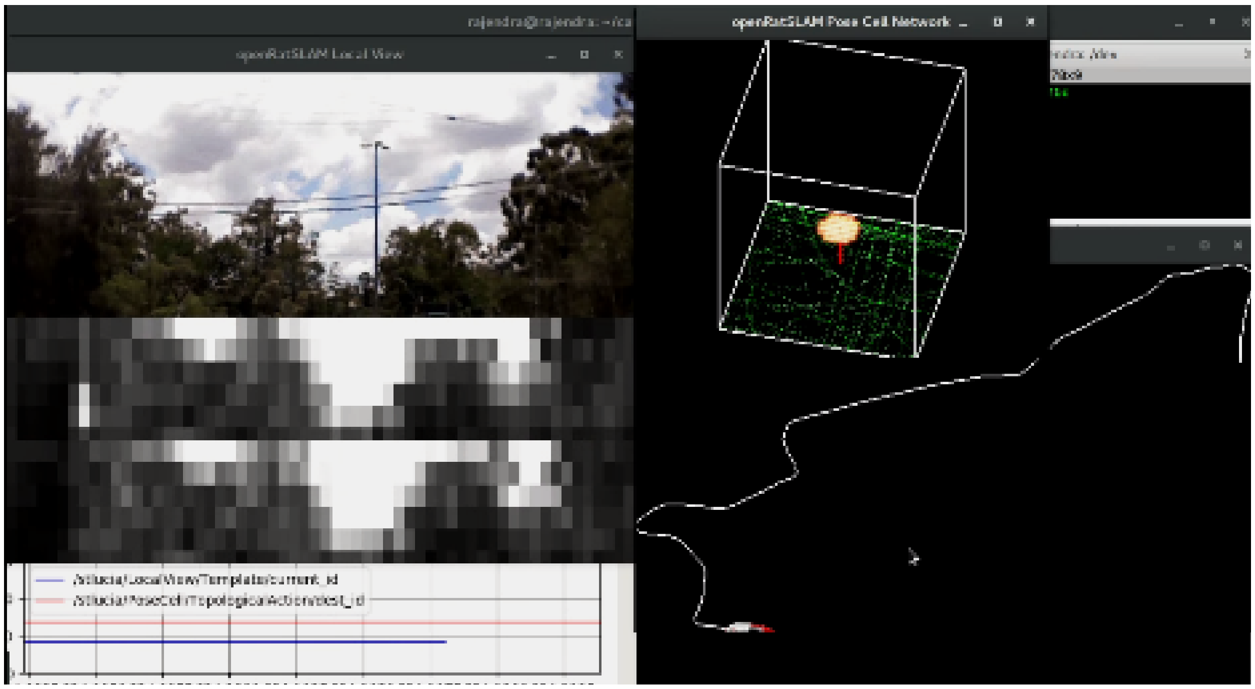


Figure 10. Setup used while mapping

3.2.3 Ratslam

[Demo](#)

RatSLAM is a bio-inspired SLAM system. Based on continuous attractor network dynamics, RatSLAM is capable of mapping by closing loops to correct odometry errors. RatSLAM is a navigation system based on the neural processes underlying navigation in the rodent brain, capable of operating with low resolution monocular image data. OpenRatSLAM is an open-source version of RatSLAM with bindings to the Robot Operating System framework to leverage advantages such as robot and sensor abstraction, networking, data playback, and visualization. OpenRatSLAM comprises connected ROS nodes to represent RatSLAM's pose cells, experience map, and local view cells, as well as a fourth node that provides visual odometry estimates.

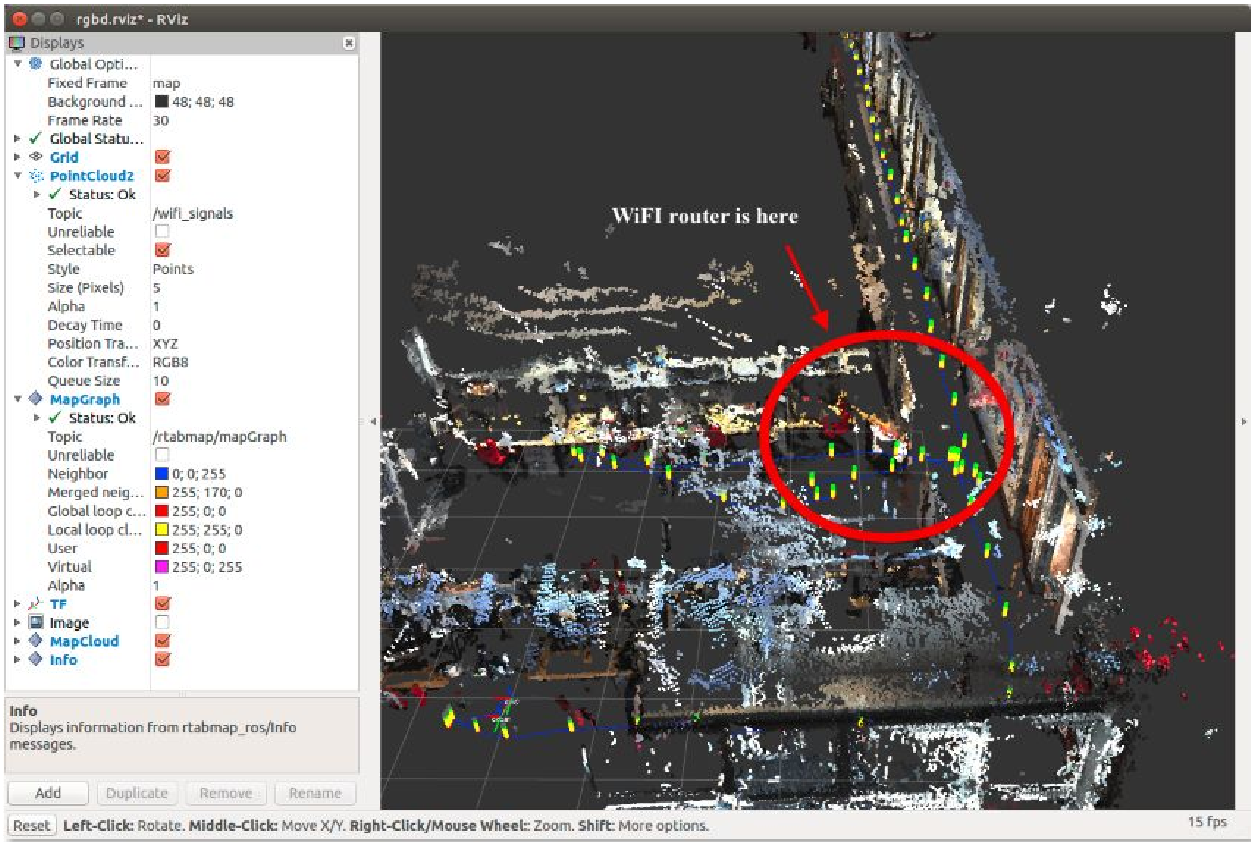

openRatSLAM Local View
openRatSLAM Pose Cell Network
/irat_red/LocalViewTemplate/current_id
/irat_red/PoseCell/TopologicalAction/dest_id
Figure 11. Ratslam demo

Implementation of OpenRatSLAM consists of four nodes.

- Pose Cell Network—This node manages the energy packet that represents pose in response to odometric and local view connections.
- Local View Cells—This node determines whether a scene given by the current view is novel or familiar by using image comparison techniques.
- Experience Map—This node manages graph building, graph relaxation and path planning.
- Visual Odometry—For image only datasets, this node provides an odometric estimate based on changes in the visual scene. The other nodes are unaware of the source of the odometric information.

3.2.4 RTAB-Map with Wifi

It is possible to use the wifi signal strength to localise robot indoors. In fact we can run RTAB-Map slam algo, one we used previously, to visualise the wifi strength in mapped environment. Here to add user data(wifi strength here) to map, rtabmap node has topics called *user_data* and *user_data_async*. The first one is used when parameter *subscribe_user_data:=true*, and it will be synchronized with other sensor data. It means that the publishing rate of this topic should be high (at least as fast as the images received). The second topic *user_data_async* is asynchronous. It should be used only when user data is published slower than the detection rate of rtabmap (default 1 Hz). The received user data will be saved in the next node created in rtabmap (if user data is published faster than nodes are created, old user data will be overwritten).



Global Opti...	
Fixed Frame	map
Background ...	■ 48; 48; 48
Frame Rate	30
✓ Global Statu...	
Grid	✓
PointCloud2	✓
✓ Status: Ok	
Topic	/wifi_signals
Unreliable	☐
Selectable	✓
Style	Points
Size (Pixels)	5
Alpha	1
Decay Time	0
Position Tra...	XYZ
Color Transf...	RGB8
Queue Size	10
MapGraph	✓
✓ Status: Ok	
Topic	/rtabmap/mapGraph
Unreliable	☐
Neighbor	🟦 0; 0; 255
Merged neig...	🟧 255; 170; 0
Global loop c...	🟥 255; 0; 0
Local loop cl...	🟨 255; 255; 0
User	🟥 255; 0; 0
Virtual	🟪 255; 0; 255
Alpha	1
TF	✓
Image	☐
MapCloud	✓
Info	✓

Info
Displays information from rtabmap_ros/Info messages.

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:** Zoom. **Shift:** More options.

Figure 12. Rtabmap with wifi

4. Robotic Manipulator

4.1 Introduction

For pick and place any object we required to know the shape and size of the objects which compose it from every point of view to be able to plan a safe trajectory. The model can be polygonal, which means that the objects are approximated by a set of polygons with or without the same shape and size. An inconvenient will be that the curves will not be smooth. This effect can be countered using curve modelling. In fact, curve modelling assigns each point on a surface a weight which will pull or push the curve trying to recreate this surface. ROS community started to develop dedicated libraries and integrate environment modelling in bigger projects. With this dynamic, the Octomap library was released in 2013 and provides a useful set of tools for octree building. 3D modelling is used to create a map of the robot's close environment. As a consequence, the position of obstacles will be known in a global frame. We will be able to correctly use path planning algorithms and avoid collisions. The octree generation requires pre-processing in order to only keep important information from the cameras.
The octree concept is a powerful way to model unknown 3D objects. An octree is not based on colour or texture.

4.2 ROS Implementation

4.2.1 Creating the realistic simulation model of uarm

As discussed previously, in urdf we describes the information about robot's joints and links. In our case, uarm swift pro, let label these as follows:

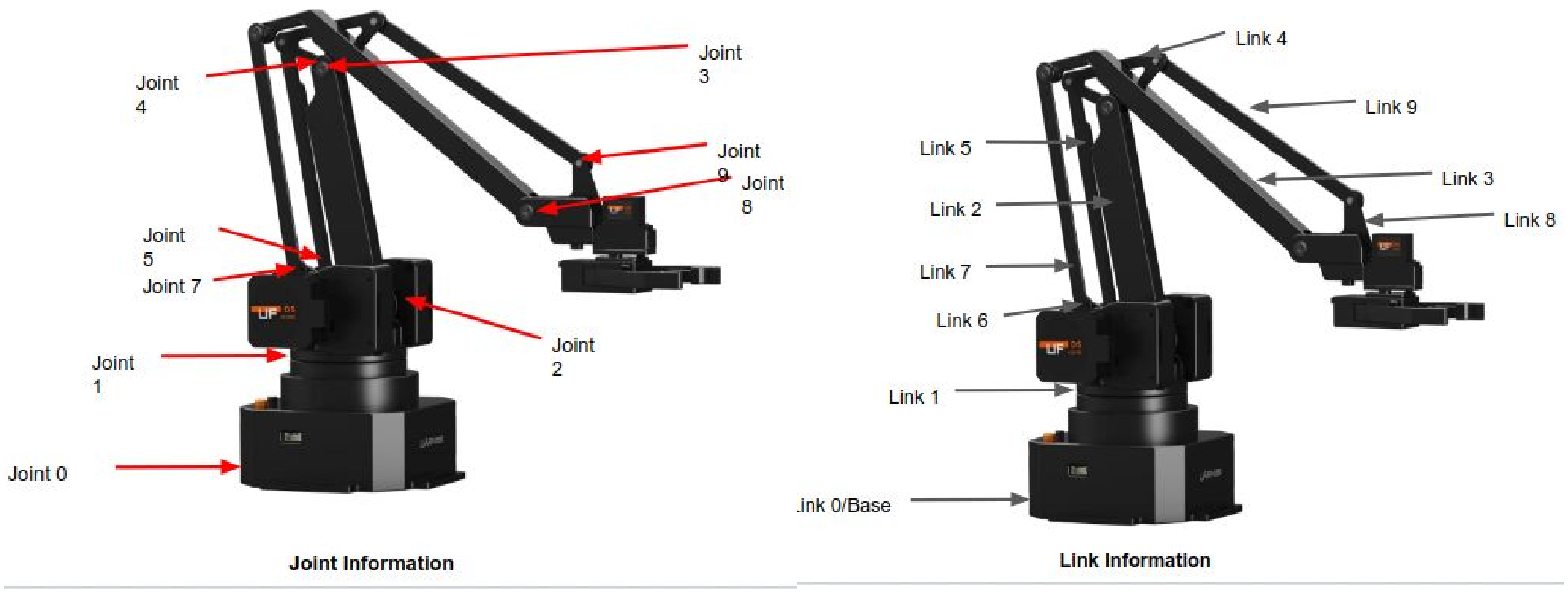
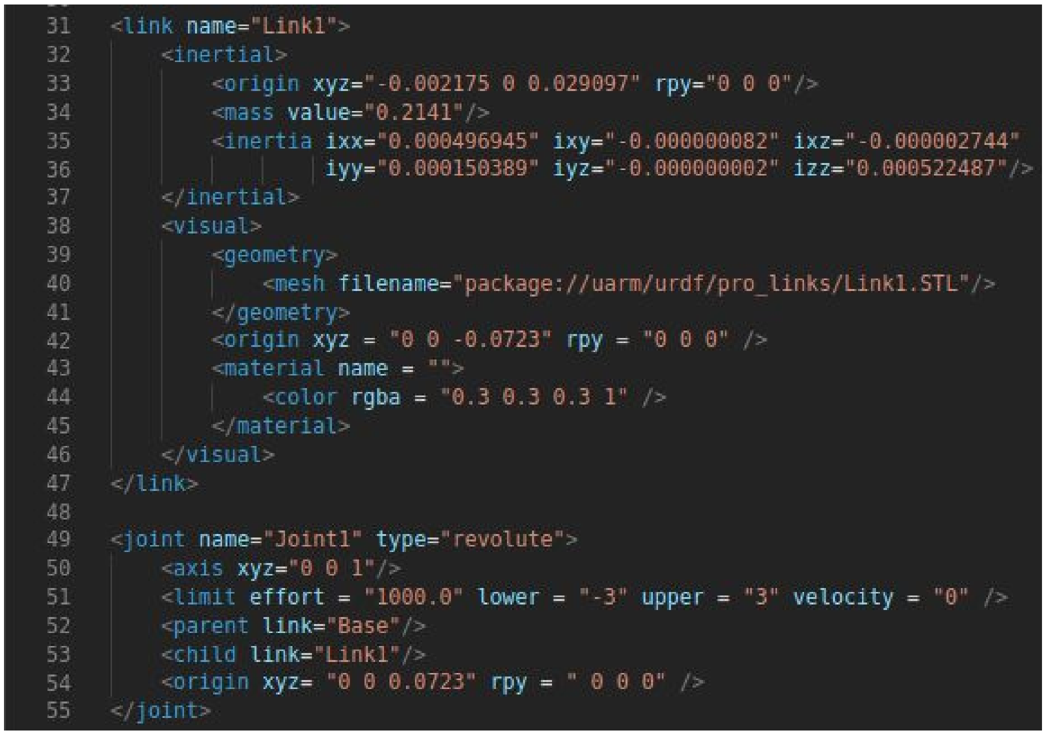


Figure 13. Uarm link and joints

So we write urdf file for our robotic arm as follows:



```xml
<link name="Link1">
    <inertial>
        <origin xyz="-0.002175 0 0.029097" rpy="0 0 0"/>
        <mass value="0.2141"/>
        <inertia ixx="0.000496945" ixy="-0.000000082" ixz="-0.000002744"
                 iyy="0.000150389" iyz="-0.000000002" izz="0.000522487"/>
    </inertial>
    <visual>
        <geometry>
            <mesh filename="package://uarm/urdf/pro_links/Link1.STL"/>
        </geometry>
        <origin xyz = "0 0 -0.0723" rpy = "0 0 0" />
        <material name = "">
            <color rgba = "0.3 0.3 0.3 1" />
        </material>
    </visual>
</link>

<joint name="Joint1" type="revolute">
    <axis xyz="0 0 1"/>
    <limit effort = "1000.0" lower = "-3" upper = "3" velocity = "0" />
    <parent link="Base"/>
    <child link="Link1"/>
    <origin xyz= "0 0 0.0723" rpy = " 0 0 0" />
</joint>
```

Figure 14. Code sample to show how to define link and joint in urdf

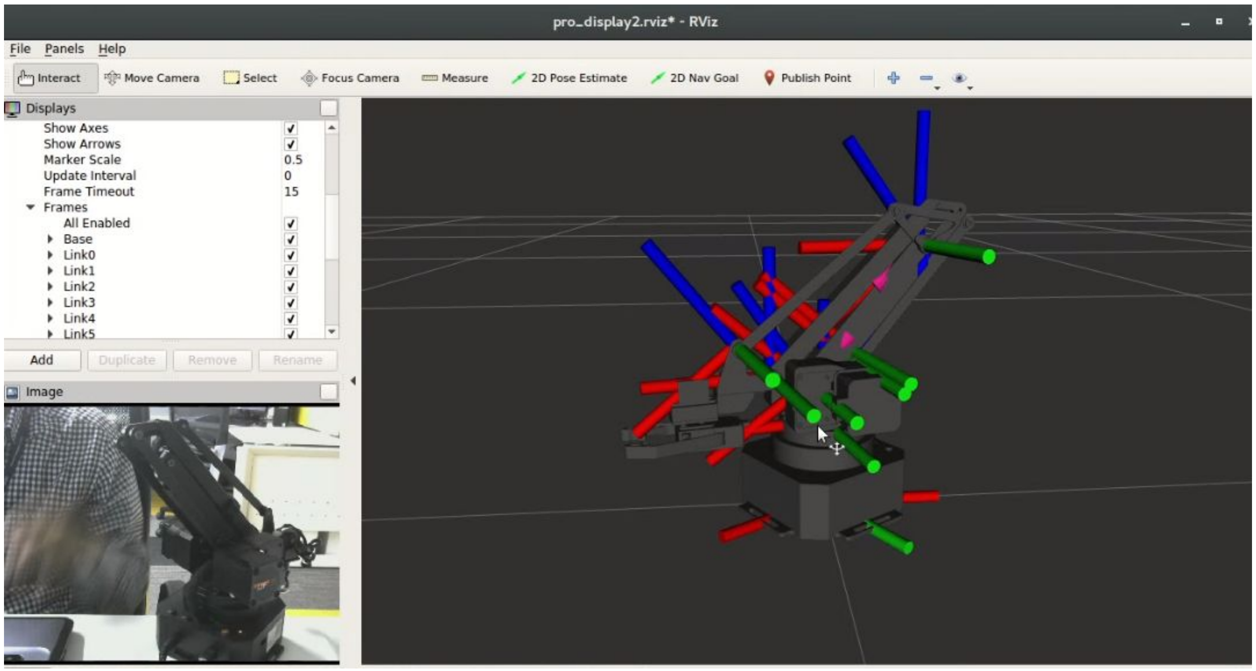
In a similar manner we write the inertia, mass,geometry,child, parent,limit and effort details for each link and joint pair.

4.2.2 Display uarm jointstate in Rviz: demo

Now we run the joint state publisher which will publish the joints states on jointState topic using above urdf file. And robot state publisher we use the JointState topic to publish the robot state on tf topic.

Hence this way we can get the frame transform between any two links.

Then we write the ros node for serial communication with uarm and which subscribe stepper motor position from motor encoders. This way we know the joints position for all the joint, which we can visualise rviz as below:


pro_display2.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Add Duplicate Remove Rename
Image

```
Show Axes          ✓
Show Arrows        ✓
Marker Scale       0.5
Update Interval    0
Frame Timeout      15
Frames
  All Enabled      ✓
  Base             ✓
  Link0            ✓
  Link1            ✓
  Link2            ✓
  Link3            ✓
  Link4            ✓
  Link5            ✓
```

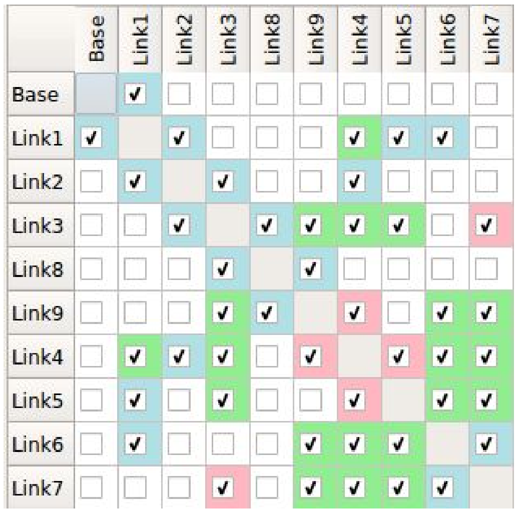
Figure 15. Displaying uarm jointstate in rviz

4.2.3 Controlling uarm jointstate

demo

We create another ros nodes which is responsible for writing the joints state to the real robot. Now we create the uarm_moveit_config package for controlling the uarm as below:

A. Optimize self collision checking

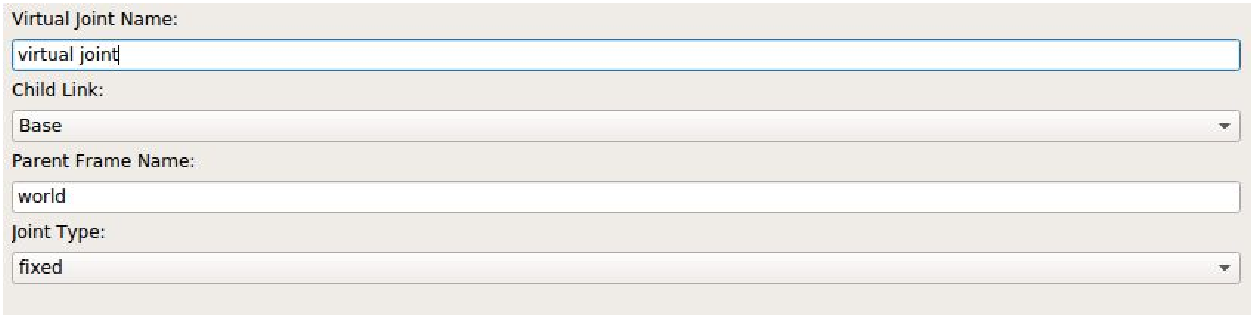


	Base	Link1	Link2	Link3	Link8	Link9	Link4	Link5	Link6	Link7
Base		✓	☐	☐	☐	☐	☐	☐	☐	☐
Link1	✓		✓	☐	☐	☐	✓	✓	✓	☐
Link2	☐	✓		✓	☐	☐	✓	☐	☐	☐
Link3	☐	☐	✓		✓	✓	✓	✓	☐	✓
Link8	☐	☐	☐	✓		✓	☐	☐	☐	☐
Link9	☐	☐	☐	✓	✓		✓	☐	✓	✓
Link4	☐	✓	✓	✓	☐	✓		✓	✓	✓
Link5	☐	✓	☐	✓	☐	☐	✓		✓	✓
Link6	☐	✓	☐	☐	☐	✓	✓	✓		✓
Link7	☐	☐	☐	✓	☐	✓	✓	✓	✓	

Figure 16. Collision checking matrix

B. Virtual joints:

Lets attach the arm to virtual fixed joint in the environment to get the appropriate transformation.



Virtual Joint Name: virtual joint

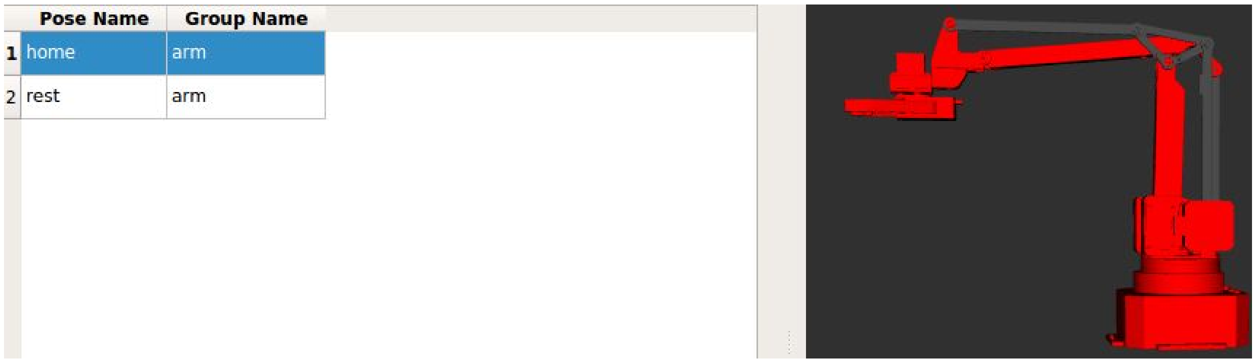
Child Link: Base

Parent Frame Name: world

Joint Type: fixed

Figure 17. virtual joints

C. Pose

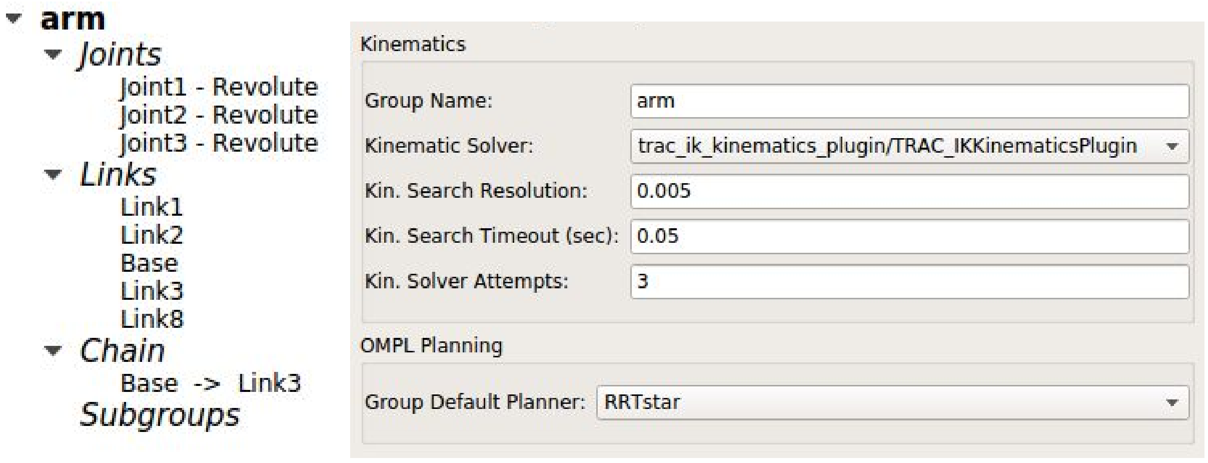


	Pose Name	Group Name
1	home	arm
2	rest	arm

Figure 18. Home pose of uarm

D. Planning group

Let create the planning group for arm, where we use trac_ik_kinematics_plugin for inverse kinematics and RRTstar planner with below configuration:



```
arm
  Joints
    Joint1 - Revolute
    Joint2 - Revolute
    Joint3 - Revolute
  Links
    Link1
    Link2
    Base
    Link3
    Link8
  Chain
    Base -> Link3
  Subgroups
```

Kinematics

Group Name: arm

Kinematic Solver: trac_ik_kinematics_plugin/TRAC_IKKinematicsPlugin

Kin. Search Resolution: 0.005

Kin. Search Timeout (sec): 0.05

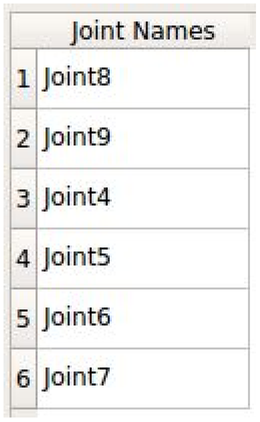
Kin. Solver Attempts: 3

OMPL Planning

Group Default Planner: RRTstar

Figure 19. Arm planning group

E. Passive joints



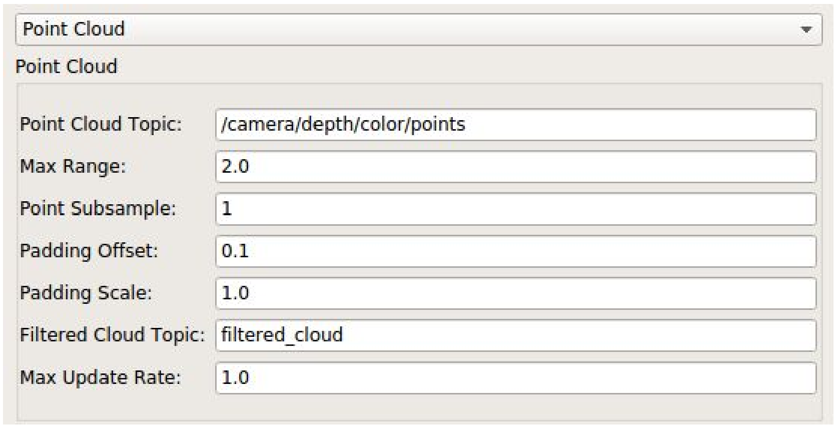
	Joint Names
1	Joint8
2	Joint9
3	Joint4
4	Joint5
5	Joint6
6	Joint7

Figure 20. Passive joints

F. Point cloud

We are subscribing the below pointcloud2 topic from realsense camera with following configuration.

Now we can do appropriate remapping of the 3d camera(realsense D435 here) topic and uarm topics to plan and execute and uarm.



Point Cloud

Point Cloud

Point Cloud Topic:	/camera/depth/color/points
Max Range:	2.0
Point Subsample:	1
Padding Offset:	0.1
Padding Scale:	1.0
Filtered Cloud Topic:	filtered_cloud
Max Update Rate:	1.0

Figure 21. Point cloud topic details

4.2.4 Perception demo

Here we will try to extract cylinder out of raw pointcloud2 data from bag file which is described in detail below:

A. Lets run the bagfile which publish pointcloud2 data.
B. Now create another node which subscribes the above pointcloud data.
C. Lets first converting sensor_msgs to pcl:PointXYZRGB format so that we can use pcl library for processing.
D. Now let's define a function passThroughFilter where we take only the data in our region of interest and discard the rest. This region of interest is here defined on max and min limit of z, cylinder can be in, to minimise future computing.
E. Now using pointcloud we'll compute the point normals and store it in cloud_normals
F. Detect and eliminate the plane on which the cylinder is kept to ease the process of finding the cylinder.
G. extracting the normals that correspond to the plane on which cylinder lies.

H. Now using cloud normals we will extract the cylinder(by extracting the pointcloud containing just the cylinder) from pointcloud data using SACSegmentation.
I. Now since we have the pointcloud containing just the cylinder, so we will compute its center point and its height and store in cylinder_params(cylinder dimension, coordinate and orientation)
J. Now we know the position and the orientation of the cylinder.

4.2.5 Pick and place object

demo

In moveit pick and place of any object can be performed in following steps:

A. We need to add the object we want to pick in planning scene environment in simulation at the coordinate, we obtained in previous perception step(cylinder coordinated).
B. Here we will add 3 cubical object planning where one of them we use to pick and place purpose and other two as platform to support the object.
C. Now we declare the gripper(similar to arm group which is use to grasp object) and object add by as non-collidable so that we can grasp the object without getting the warning about false collisions.
D. For successful pick and place we have to declare the pre-grasp, post-grasp, grasp approaches to able to safely pick object without it being displaced from its position or damaged.
E. In pre-grasp approach we define from which side arm should grasp the object and how far from object we should open the gripper etc.
F. Post-grasp describe how should we move after grasping the object. E.g if we are picking the object from the table than we should not -z axis because that way we'll hit the table.
G. Now we will launch panda arm simulation.(Panda is name of another robotic arm which has more DOF than uarm, the one we were using previously, so we can plan the movement without plan getting aborted!
H. So once the simulation is up we will run our main pick and place node is written while keeping all points we discussed above in mind
I. So firstly object will be added in planning scene and we will approach following planned trajectory and pre-grasp approach.
J. Now once we at object position, we'll attach the object to gripper group and remove from planning environment(so that now object we with gripper as it is part of it) and close the gripper and plan another trajectory where we want to place this object and execute this trajectory.
K. Once we are final position we will open gripper and detach an object and add it back to the planning scene and open gripper. This complete pick and place of this object. So now declare back gripper and this object to collidable.

5. Sensor fusion:

5.1 Commonly used sensor with slam:

5.1.1 Gyroscope

Gyroscopes have been widely used for military and civilian purposes in the last century. Not only for aeronautical purposes but also by the marine. The traditional gyroscopes used to be mechanical. However, what is being used in many applications nowadays are MEMS gyroscopes (micro-electro-mechanical systems), which are being used in cell phones, robotic projects, and for military purposes. Gyroscopes, in general, can be described by the formula of Coriolis force

$$F_c = -2 * m * (\omega * v)$$

As can be seen from the above equation the Coriolis force is depending on the mass m of an object, the angular velocity ω as well as the velocity v of the object.

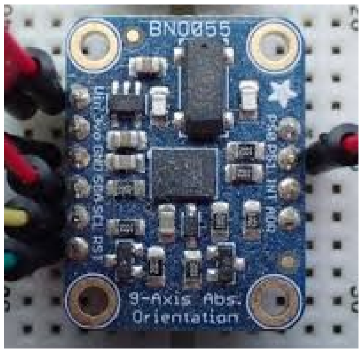


Figure 22. BNO055

This object is a part of the gyroscope and the mass of the object as well as the velocity of the object are known. Given that the object rotates, and by having information about the mass and velocity of the object the angular velocity can be detected.

We'll write ros node which will publish data as ros compatible message which is of the following format:

std_msgs/Header header
geometry_msgs/Quaternion orientation
float64[9] orientation_covariance
geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance
geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance

5.1.2 Wheel encoders

Wheel encoders are being used to provide the microcontroller with information about the velocity of the vehicle. There are several different types of wheel encoders based on different technologies, such as optical wheel encoders, mechanical as well as magnetic wheel encoders just to name a few. In magnetic wheel encoder use hall effect. The Hall effect sensor outputs a high voltage level whenever the magnetic field surrounding it is sufficiently strong, (a magnet passing by). The pulse outputted by the Hall effect sensor is being registered by the microcontroller. By counting the number of pulses generated since the previous sampling instance, the microcontroller is able to estimate the speed of the vehicle. We count pulse frequency to find the speed of which and the velocity of robot. We create a ros node which will read this data from sensor and publish geometry_msgs/Twist msg which is of the following format:

Figure 23. Wheel encoder

geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular

5.1.3 Laser range finder

Laser Range Finders (LIDAR) can be used to measure the distance to objects in the vicinity of the vehicle. A laser range finder consists of a laser emitter and receiver and is sometimes mounted on a rotating motor in order to measure distances in every direction of the vehicle. The basic operation principle is that a short pulse of light is emitted, reflected on an object, and then received. The time between the emitting and receiving can then be used to calculate the distance d to the object. We create a ros node which will read this data from sensor and publish sensor_msgs/LaserScan msg which is of the following format:

```
std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

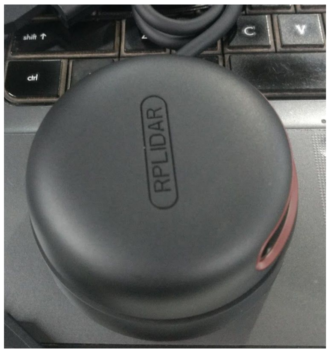


Figure 24. Rplidar A208

5.1.4 3D camera:

I used realsense d435 to get the pointcloud2 data which of the following format:

```
std_msgs/Header header
uint32 height
uint32 width
sensor_msgs/PointField[] fields
bool is_bigendian
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```

Figure 25. Realsense D435

5.1.5 GPS :

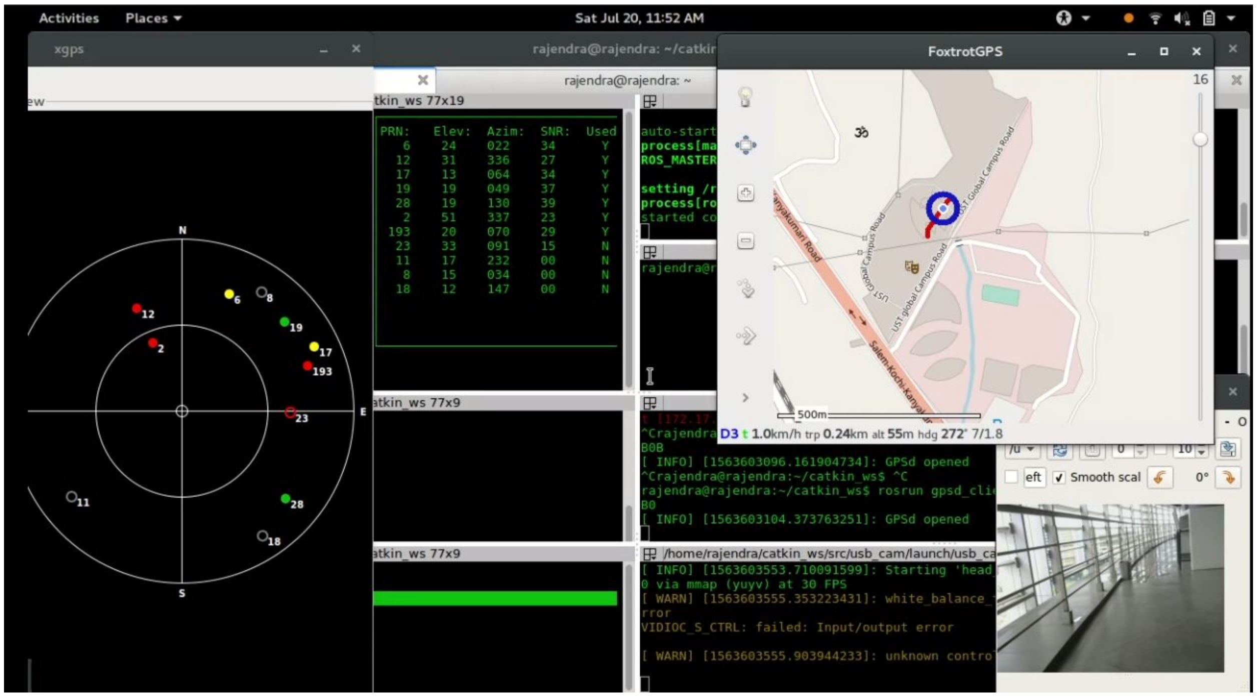
demo

I used G STAR IV and garmin glo 2 sensor. Out of which G STAR IV was usb compatible and was used to publish gps data with help gzclient ros node. Here Gps data is of sensor_msgs/NavSatFix format which is as below:

```
uint8 COVARIANCE_TYPE_UNKNOWN=0
uint8 COVARIANCE_TYPE_APPROXIMATED=1
uint8 COVARIANCE_TYPE_DIAGONAL_KNOWN=2
uint8 COVARIANCE_TYPE_KNOWN=3
std_msgs/Header header
sensor_msgs/NavSatStatus status
float64 latitude
float64 longitude
float64 altitude
float64[9] position_covariance
uint8 position_covariance_type
```

Figure 26. Garmin glo 2

Figure 27. GStar IV GPS



```
PRN:   Elev:  Azim:  SNR:  Used
  6     24     022    34     Y
 12     31     336    27     Y
 17     13     064    34     Y
 19     19     049    37     Y
 28     19     130    39     Y
  2     51     337    23     Y
193     20     070    29     Y
 23     33     091    15     N
 11     17     232    00     N
  8     15     034    00     N
 18     12     147    00     N
```

```
auto-start
process[ma
ROS_MASTER

setting /r
process[ro
started co
```

```
rajendra@r
```

```
^Crajendra
B0B
[ INFO] [1563603096.161904734]: GPSd opened
^Crajendra@rajendra:~/catkin_ws$ ^C
rajendra@rajendra:~/catkin_ws$ rosrun gpsd_clie
B0
[ INFO] [1563603104.373763251]: GPSd opened
```

```
/home/rajendra/catkin_ws/src/usb_cam/launch/usb_ca
[ INFO] [1563603553.710091599]: Starting 'head
0 via mmap (yuyv) at 30 FPS
[ WARN] [1563603555.353223431]: white_balance_
rror
VIDIOC_S_CTRL: failed: Input/output error

[ WARN] [1563603555.903944233]: unknown contro
```

Figure 28. Using Gps with ros, to get navsat data

5.2 Sensor fusion using robot_localisation:

For fusing various sensor data I used robot_localisation package. It is a collection of state estimation nodes, each of which is an implementation of a nonlinear state estimator for robots moving in 3D space. It contains two state estimation nodes, ekf_localization_node and ukf_localization_node. In addition, robot_localization provides navsat_transform_node, which aids in the integration of GPS data.

All the state estimation nodes in robot_localization share common features, namely:

- Fusion of an arbitrary number of sensors. The nodes do not restrict the number of input sources. If, for example, your robot has multiple IMUs or multiple sources of odometry information, the state estimation nodes within robot_localization can support all of them.
- Support for multiple ROS message types. All state estimation nodes in robot_localization can take in nav_msgs/Odometry, sensor_msgs/Imu, geometry_msgs/PoseWithCovarianceStamped, or geometry_msgs/TwistWithCovarianceStamped messages.
- Per-sensor input customization. If a given sensor message contains data that you don't want to include in your state estimate, the state estimation nodes in robot_localizationallow you to exclude that data on a per-sensor basis.
- Continuous estimation. Each state estimation node in robot_localization begins estimating the vehicle's state as soon as it receives a single measurement. If there is a holiday in the sensor data (i.e., a long period in which no data is received), the filter will continue to estimate the robot's state via an internal motion model.

<u>All state estimation nodes track the 15-dimensional state of the vehicle:</u>
$(X,Y,Z,roll,pitch,yaw,\dot{X},\dot{Y},\dot{Z},\dot{roll},\dot{pitch},\dot{yaw},\ddot{X},\ddot{Y},\ddot{Z})$

- ekf_localization_node is an implementation of an extended Kalman filter. It uses an omnidirectional motion model to project the state forward in time, and corrects that projected estimate using perceived sensor data.

- navsat_transform_node takes as input a nav_msgs/Odometry message (usually the output of ekf_localization_node or ukf_localization_node), a sensor_msgs/Imu containing an accurate estimate of your robot's heading, and a sensor_msgs/NavSatFix message containing GPS data. It produces an odometry message in coordinates that are consistent with robot's world frame. This value can be directly fused into state estimate.

Integrating GPS:

1. Run one instance of a robot_localization state estimation node that fuses only continuous data, such as odometry and IMU data. Set the world_frame parameter for this instance to the same value as the odom_frame parameter. Execute local path plans and motions in this frame.
2. Run another instance of a robot_localization state estimation node that fuses all sources of data, including the GPS. Set the world_frame parameter for this instance to the same value as the map_frame parameter

6. ROS 2

ROS started on November 2007. But recently new version ROS comes up, which is known as ROS 2. There are few use cases ros doesn't meet so far. So handle this ROS 2 was launched. These mentioned use cases are as follows:

- Teams of multiple robots: while it is possible to build multi-robot systems using ROS today, there is no standard approach, and they are all somewhat of a hack on top of the single-master(i.e master slave) structure of ROS. But know we want peer to peer communication in true sensor, where no one being master and slave.
- Small embedded platforms: There is a need to make even small computers, including "bare-metal" micro controllers, to be first-class participants in the ROS environment, instead of being segregated from ROS by a device driver.
- Real-time systems: There is a need to support real-time control directly in ROS, including inter-process and inter-machine communication
- Non-ideal networks: Its required that ROS behave as well as is possible when network connectivity degrades due to loss and delay, from poor-quality WiFi to ground-to-space communication links.
- Prescribed patterns for building and structuring systems:Along with maintaining underlying flexibility that is the hallmark of ROS, there is a need to provide clear patterns and supporting tools for features such as life cycle management and static configurations for deployment. So know now ROS 2 node have life cycle.

New technologies incorporated in ros 2 are:

- Zeroconf : Zero configuration networking refers to several protocols and techniques that are used together to create an IP network with no special configuration servers or manual operator intervention. Zeroconf does not require the user to set up DNS, DHCP, or manually configure the computer's network settings.
- Protocol Buffers : These are a flexible, efficient, automated mechanism for serializing structured data. E.g sending rosmsg.
- ZeroMQ : It is a high-performance asynchronous messaging library, aimed at use in distributed or concurrent applications. It provides a message queue, but unlike message-oriented middleware, a ZeroMQ system can run without a dedicated message broker. E.g multiple node can send and receive msg from same topic
- Redis : It is an in-memory data structure project implementing a distributed, in-memory key-value database with optional durability. Redis supports different kinds of abstract data structures, such as strings, lists, maps, etc.
- WebSockets : It is a computer communications protocol, providing full-duplex communication channels over a single TCP connection

7. Embedded boards for real robot:

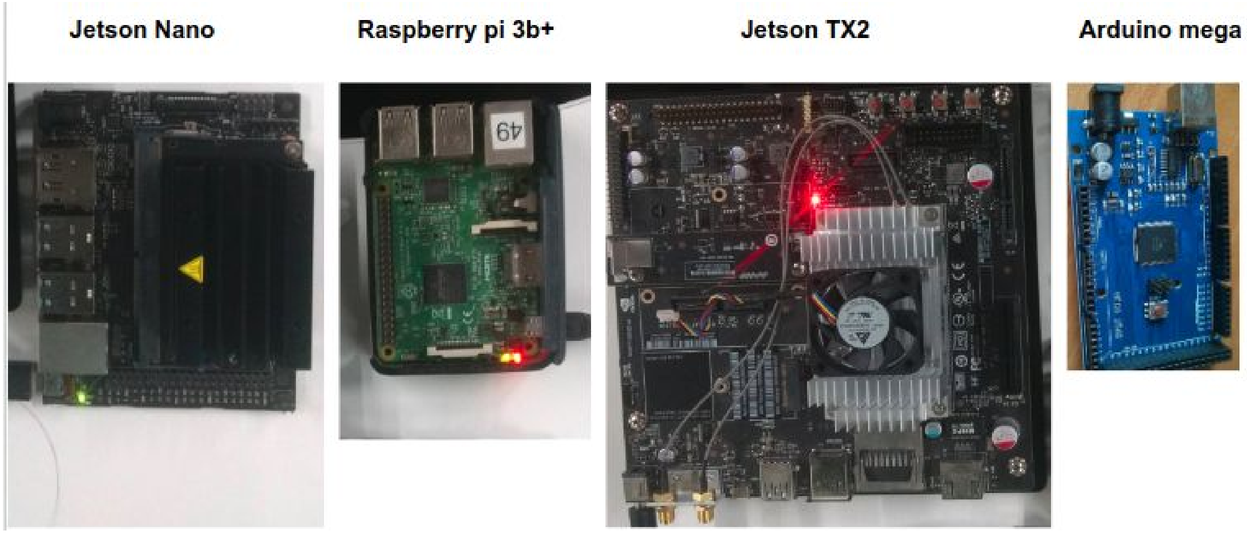


Figure 29. Various embedded board for real robot

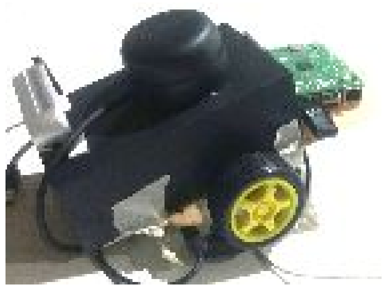


Figure 30. RC car slam

I used to raspberry pi 3b+ to run the ros on the rc car, where we were publishing the pointcloud2 and laserscan data from realsense camera and rplidar respectively. This data was used to subscribed by the ros node running in laptop(rosmaster) which was undertaking all the heavy computation work.
We are using Arduino Atmega16 as slave node on rc car which responding for driving the wheel motors.

I setup ros to get the sensor data in Jetson TX2 and Nano, but finally I used raspberry pi in rc car for the same.

8. Swarm robotics

8.1 Introduction

Swarm robotics is an approach to the coordination of multiple robots as a system which consists of large numbers of mostly simple physical robots. It is supposed that a desired collective behavior emerges from the interactions between the robots and interactions of robots with the environment.

8.2 Swarm robotics using drone

Smart India Hackathon is national level competition. My team GreatEagle got selected for swarm related problem. I worked on this problem beforehand and I get a chance to continue that during my intern. Below I my work in brief:

8.3 Problem Statement

The basic objective of the challenge is to develop an algorithm to control 2 drones synchronously and move the pair from one point to another maintaining the attitude of each of the drones relative to one another. The two drones have to start from a marked starting point, at a specific relative distance and attitude to each other. They have to execute a series of control commands, navigate a predefined path and then land at the end point, maintaining the same relative distance and attitude throughout the flight. The drones have to be able to estimate their location wrt each other and the world(transmitter) , compare the same and be able to determine their true position. Feedback from visual sensors or any other sensors available on a standard drone can be used. The entire operation needs to be performed in a fully autonomous fashion. The usage of way point markers or other markers to define location feedback is allowed for simplicity.

8.4 ABSTRACT IDEA

demo

Kindly find full report here, below I'm presenting the abstract idea of our approach on which I worked during my intern.

Since the precise application for problem statement was not known, we proposed two solutions, one for indoor and the other for outdoor.

For indoor drone navigation, we're using markers for localization of checkpoints of the path, obstacles and on the drones to locate them in the frame of an overhead camera which is watching this whole setup from the top. We used Aruco markers with different ids for obstacles and checkpoints of path. We also placed modified Whycon markers on the drones to detect their position and yaw. Since the overhead camera is able to locate each of the markers in its frame of reference, both drones can now navigate on the predefined path synchronously while avoiding obstacles. PID control algorithm is used for drones to navigate between any two coordinates. We have to find such a path which can avoid obstacles and also navigate both drones maintaining their relative distance and attitude. For the same, we used OMPL on ROSand simulated the same using a software called VREP. Here, we grouped both the drones and considered them as an object where only drones and obstacles were set as collidable. We use the OMPL algorithm to plan a path for midpoint of the group avoiding obstacles. Now, to get the individual path for both drones, firstly divided the midpoint path in some point called way-points. Now we found the initial offset of both drones from the mid-point in vector form, added it to each coordinate of midpoint path way-point. So, this way we are able to find the two seperate paths for both the drones where distance between the drones is being maintained. So the only thing to be done now is to run both the drones on their corresponding path synchronously. We will give the way-points to the drone simultaneously. Suppose we have given the first pair of corresponding way-points to the drones. The next pair of way-points will be given only after both the drones reach their corresponding previous way-points properly. A point is said to be properly reached only if its error in all three axes is less than some given constant value.

For outdoor navigation approach, we proposed a different solution. We are using GPS module and front-facing cameras in both drones using which we are able to locate both drones and obstacles w.r.t the ground frame, hence enabling us to navigate both drones between two positions synchronously while avoiding obstacles. For localization, we used GPS which was integrated with IMU using Kalman filter for more precise location data. Here we used waypoint GPS Navigation which allows a drone to fly on its own with its flying destination or points pre-planned and configured into the drone's remote control navigational software. Waypoints to be selected will be of the midpoint path. All the waypoint coordinates will be stored and the next waypoint will be checked for. Depending on the UAV's position, straight flight or turn with a constant angle (depending on the path given and presence of obstacles) command will be given. When it reaches there, the same procedure is followed for the next destination point with similar algorithm as used in indoor navigation. To avoid obstacles, we used a different approach. We will use the obstacle-restriction method like VFH (Vector Field Histogram) algorithm to avoid an obstacle and move in the available window. Since we consider both drones together making them into a single object where only drones are collidable, the path will be planned for the midpoint of this combined object. Using the path of the midpoint, we can derive the path of individual drones similar to the case of indoor navigation. Since the relative distance between these derived paths will be constant (because of being derived using the same midpoint path), both drones will be able to maintain their relative distance and attitude. For efficiently avoiding obstacles in 3D, SLAM algorithms like LSD-SLAM will be used. As we have used ROS to control the drones, they will be connected to a laptop wirelessly. A Master node running in the laptop will perform all the computation, path planning and will also control the drones by publishing to control topics. Both the drones will publish the sensor data to their corresponding topics and will subscribe to control topics for navigation. We will also consider using some other method for combining the indoor and outdoor approaches depending on the given situation.

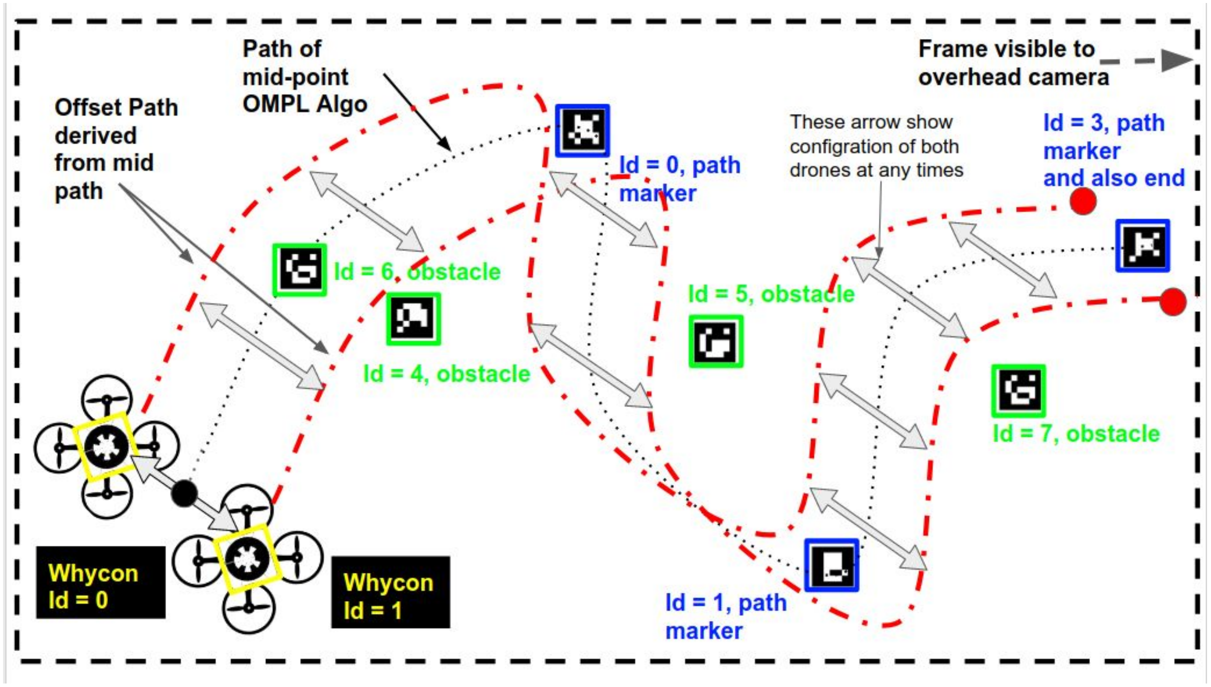
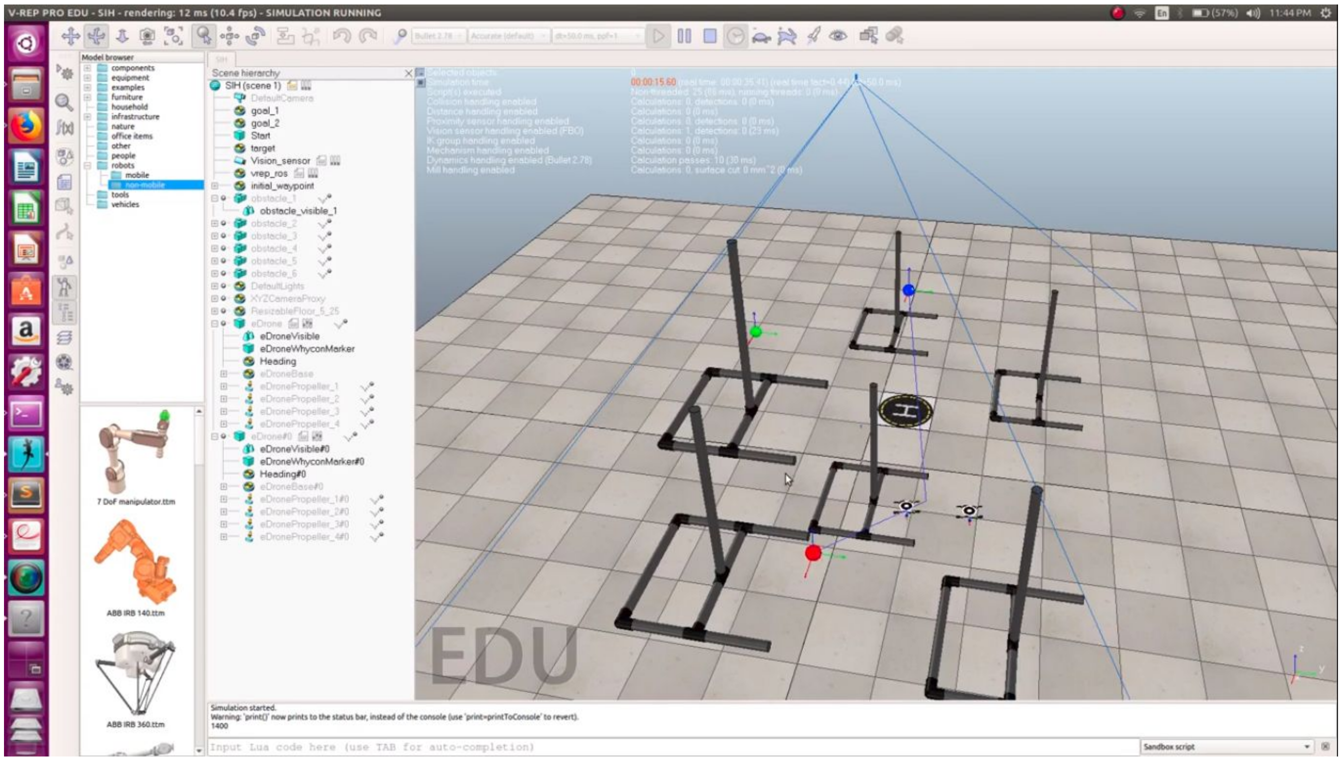

Path of mid-point OMPL Algo
Frame visible to overhead camera
Offset Path derived from mid path
These arrow show configration of both drones at any times
Id = 0, path marker
Id = 3, path marker and also end
Id = 6, obstacle
Id = 5, obstacle
Id = 4, obstacle
Id = 7, obstacle
Whycon Id = 0
Whycon Id = 1
Id = 1, path marker

Figure 31. Indoor navigation using waypoint markers


V-REP PRO EDU - SIH - rendering: 12 ms (10.4 fps) - SIMULATION RUNNING
Model browser
Scene hierarchy
EDU

```
Simulation started.
Warning: 'print()' now prints to the status bar, instead of the console (use 'print=printToConsole' to revert).
1400
```

Figure 32. V-REP simulation of two drones moving in sync

FUTURE SCOPE AND PLAN

I'm extremely enthusiastic about robotics. I can see how new advancement in drones and swarm robotics going to affect our lives in the upcoming 1-2 decades. We all know AI is the next big thing. With AI, robotics industry going be fully revolutionised, as it enables robot to accomplice cognitive task which wasn't possible before. Only thing I was a bit worried while building any of these robots using middleware like ROS was its reliability, as it didn't meet the industry standard yet. But thanks to the ROS2 which handle some of these issues now. As autonomous driving is rapidly becoming the next major challenge in the automotive industry, the problem of SLAM has never been more relevant than it is today.
I will be continuing to learn more about slam, build upon on concepts I learnt during this internship. Indeed I have already taken up slam as my final year project, and excited to learn more about it.

SUMMARY

This internship was very fruitful to me. During this period I had the opportunity to learn and work on a great variety of projects, which has been very enriching. I got to learn a few topics in depth, which I was previously working on. However, I wasn't able to bring conclusion to some projects I worked on. In this section I will quickly summarize what has been done, and what remains to be done for all these projects. Learning about indoor localisation, by being in the lab and working with other colleagues, I have gained a good overview of the domain. I acquired a good understanding of the various slam algorithms such as rtabmap, gmapping and ratslam and also implemented these using sensors like monocular camera, lidar, 3d camera. While I made good progress in mapping and localisation, there is still a lot of room for navigation which requires good knowledge of both hardware and software domain.
Meanwhile these work, the project which really caught my eye is mobile manipulator and possibility of it's endless application. I'm excited to learn more about it in the coming time.
To conclude, I think that internship was very beneficial to me as I learnt a lot about the field, and it introduced me to the industrial level of programming and honed my skills.

CERTIFICATE

UST Global®

July 22, 2019

TO WHOMSOEVER IT MAY CONCERN

This is to certify that Rajendra Singh (ID: 102370) from IIT Palakkad has completed internship in UST Global from May 03, 2019 to July 22, 2019.

Rajendra Singh's performance was good during this period and he/she has successfully completed his/her internship training at UST Global on Path planning and robot manipulation.

UST Global wishes his/her all the success for his/her future endeavours.

For US Technology International Pvt. Ltd

[illegible]

Anwar Khan Sheffi
Director – Human Resources

US Technology International Pvt. Ltd. #721, Nila, Technopark, Trivandrum-695 581, India, T: 91.471.4040000, F: 91.471.2527276, www.ust-global.com
CIN: U72200KLI999PTC013385

References

1. robot_localization, roswiki
2. RTABMAP SLAM, http://introlab.github.io/rtabmap/
3. Thrun, Burgard, Fox: Probabilistic Robotics, MIT Press, 2005, website
4. Springer Handbook on Robotics, Chapter on Simultaneous Localization and Mapping
5. Schoen and Lindsten: Manipulating the Multivariate Gaussian Density, 2011, pdf
6. Welch and Bishop: Kalman Filter Tutorial, 2011, pdf
7. Julier and Uhlmann: A New Extension of the Kalman Filter to Nonlinear Systems, 1995, pdf
8. Thrun, Liu, Koller, Ng, Ghahramani, Durrant-Whyte: Simultaneous Localization and Mapping With Sparse Extended Information Filters, 2004. pdf
9. Eustice, Walter, Leonard: Sparse Extended Information Filters: Insights into Sparsification, IROS, 2005. pdf
10. Montemerlo, Thrun, Kollar, Wegbreit: FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem, 2002, pdf
11. Montemerlo, Thrun: Simultaneous Localization and Mapping with Unknown Data Association Using FastSLAM, 2003, pdf
12. Grisetti, Stachniss, Burgard: Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters, 2007,pdf
13. Stachniss, Grisetti, Burgard, Roy: Analyzing Gaussian Proposal Distributions for Mapping with Rao-Blackwellized Particle Filters, 2007, pdf
14. Madsen, Nielsen, Tingleff: Methods for Nonlinear Least Squares Problems, 2004, pdf
15. Grisetti, Kuemmerle, Stachniss, Burgard: A Tutorial on Graph-Based SLAM, 2010, pdf
16. Grisetti, Kuemmerle, Stachniss, Frese, Hertzberg: Hierarchical Optimization on Manifolds for Online 2D and 3D Mapping, 2010, pdf
17. Olson, Agarwal: Inference on Networks of Mixtures for Robust Robot Mapping, 2013, pdf
18. Agarwal, Tipaldi, Spinello, Stachniss, Burgard: Robust Map Optimization Using Dynamic Covariance Scaling, 2013, pdf
19. Olson: Recognizing Places using Spectrally Clustered Local Matches, 2009, pdf
20. https://link.springer.com/article/10.1007%2Fs10514-012-9317-9
21. http://ipvs.informatik.uni-stuttgart.de/mlr/wp-content/uploads/2019/01/08-MobileRobotics_WS1819.pdf
22. http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/