University of Siegen
Faculty of Science and Technology
Department of Electrical Engineering and Computer Science
Chair of Automatic Control Engineering

Student Work

3D SLAM-Based Navigation of an Autonomous Mobile Robot Using Kinect Camera and ROS

by

Manoj Venkatarao Sanagapalli
Matr.-nr.: 1383616

B.Tech., Acharya Nagarjuna University, 2015

October 2019

Declaration

I hereby declare that this thesis entitled “**3D SLAM-Based Navigation of an Autonomous Mobile Robot Using Kinect Camera and ROS”**, is the result of my own work and it has not been submitted for any other degree or other purposes. Further, I have faithfully and accurately cited all sources that assisted me throughout my Student work, including other researchers‘ work published or unpublished.

I am aware of the consequences of not recognizing any sources used in the respective work and it is considered as fraud and plagiarism.

…………………………………………………..

Manoj Venkatarao Sanagapalli

Matr.-nr.: 1383616

October 2019

Abstract

3D SLAM-Based Navigation of an Autonomous Mobile Robot Using Kinect Camera and ROS

by

Manoj Venkatarao Sanagapalli
Matr.-nr.:1383616

Submitted to the Department of Electrical Engineering and Computer Science
on October 2019 in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Mechatronics

Robots are evolving from powerful stationary mechanical systems to sophisticated, mobile platforms to address a broad range of automation needs. Autonomous mobile robots (AMR) became the center of robotic innovations in the present world because of its widespread usage in different areas like industrial, military, domestic and entertainment appliances. To cope with the high functional expectations and perform desired tasks, mobile robots should be capable of sensing the environment and make real-time decisions. Designing and developing such robot systems is challenging and an area worth to study. The main focus of this project is to do indoor SLAM (Simultaneous Localization And Mapping) using an RGB-D sensor and navigate autonomously in the mapped environment. For this, a ROS based Teleoperated differential drive robot (TOM3D) is used. The robot is designed and developed by chair of automatic control and engineering, University of Siegen. Kinect V2 Camera is mounted on it. It also has Laser Scanner, wheel encoders and Inertial Measurement Unit (IMU). The low-level controller (Arduino board) of the TOM3D controls robot behavior based on commands given by computer (high-level controller). It also updates the robot state by providing encoder and IMU feedback.

First, mapping is done with manually controlling the robot in a known environment (i.e. static environment). Furthermore, ROS navigation stack is used to autonomously navigate the robot. Where a target is given to the robot in the environment and an optimum path is

generated which is then used to navigate the robot autonomously to the desired target. The robot was able to get the map of the environment successfully and navigate to the set goal while avoiding all the mapped obstacles.

Table of Contents

List of Figures

List of Tables

1 Introduction

Robots are found everywhere from lawnmowers and pool cleaners to space vehicles. Among these, mobile robots received a special emphasis on research to extend their capabilities to navigate in complex environments. Their usage in diversified fields (e.g. industry, mining, space explorations, underwater missions etc.) has created various challenges for them such as size of the environment, noise in perception etc. To overcome those challenges, the robot must have a sensing mechanism to perceive the environment as well as some sort of intelligence for interpreting the data to make decisions.

In mobile robots, planning a path in the environment to reach a particular goal is a mandatory function and is termed as navigation. To add more autonomy to robots to navigate autonomously without human interaction is a complex task and requires a wide variety of sensors, algorithms, and techniques. The SLAM-based autonomous navigation system is comprised of three sub-parts namely Mapping-"where is the robot going?", Localization-"where is the robot" and path planning-"How should the robot go there?". Robot localization can be complex task because of the uncertainties caused in sensing as the measurements taken from real-time sensors can go noisy and problematic. To resolve this problem, several methodologies like Beacons, landmarks, and route-based methods as well as algorithms based on probabilistic approaches like Markov's, particle filter localization can be used.

In autonomous navigation, sensor data is very crucial. It should be free of run-time errors and noise. Sensors are selected based on the type of environment and the desired type of the map. Common sensors are sonars, digital cameras, and laser scanners.

The development of such an autonomous navigation system requires working simultaneously in several fields which requires a lot of effort and time. Fortunately, there are many open-source RDE's available such as Robot Operating System (ROS), Robot Open Platform (ROP), Microsoft Robotics Developer Studio, etc. In this project, ROS framework is used as it provides a wide variety of open-source drivers and algorithms.

1.1 Motivation

The main motivation of this project is to enhance Tele Operated Machine Robot (TOM3D-Robot) navigation system to autonomously navigate using visual SLAM and ROS framework. The differential drive robot is developed by the chair of automatic control and engineering, University of Siegen.

1.2 Thesis Outline

The report is structured into the following chapters:

Chapter 1 provides a general overview of the project area and the motivation behind the project undertaken.

Chapter 2 introduces the background for all algorithms and methodologies used in this project. It sheds the light on different aspects of autonomous navigation such as mapping, localization and SLAM techniques used in this project. An emphasis on the ROS framework as well as packages used during development of the navigation system are then introduced.

Chapter 3 describes the hardware used in this project, including the communication between the various hardware components.

Chapter 4 presents a comprehensive explanation of the overall navigation system implementation. This involves handling of sensor data through ROS packages and the implementation of different nodes to obtain the desired system.

Chapter 5 provides a detailed discussion of the work and the results obtained.

2 Background

2.1 Autonomous Navigation

Autonomous navigation is a core capability of any robot that performs high-level tasks involving mobility. A robot should capture information about the environment [1]. It is essential for the robot to visualize the whole environment and must know its own position in the environment. The capability to visualize and localize in the environment is a significant prerequisite for navigating autonomously to a given goal.

2.1.1 Mapping

“*Given the robot’s position and sensor measurements, what are the sensors seeing?*” – [2]

As a first step, the robot should build a visual representation of the surroundings, which can be used to determine its own location and plan paths. Almost all the new indoor mobile robots use environment maps for navigation [3]. Although robots operate in a three-dimensional environment, most of the algorithms used for mapping takes only 2D view of the environment. Since the robot movement is in the 2-D plane, a 2-D map is sufficient for most of the applications. Nevertheless, 3-D mapping has its own advantages. 3-D maps process more information compared to that of 2-D. Thus, 3-D maps provide less ambiguity in the results while mapping different areas. Sensors are attached to robot base which can perceive the environment and thus creating a map. Vision-based sensors like RGB-D cameras, range-based sensors like Sonar, Laser, Infrared and GPS can be used to accomplish the mapping task.

Major challenges in mapping are:

1. Measurement noise from the sensors accumulate over time and lead to errors in the final map of the environment.
2. Correspondence or data association problem. If a physical object is measured by a sensor at two different times, the measurements should be associated properly.
3. Incorporating dynamic changes of the robot environment.

The most prominent approaches to mapping are probabilistic mapping and feature-based mapping. Shadow based and direct mapping are used to create a map of unstructured environments.

Probabilistic methods are used over the last few years to solve the uncertainties involved in the process of building a map by mobile robots [4]. Another method is suggested by [5] which closely resembles EM algorithm (Expectation maximization) which is very adaptable for estimating maximum likelihood in 3D or 2D space. In the Expectation step (E-Step), the robot computes probabilities for the robot's poses at the various points throughout time, based on the currently best available map. In the M-step, the robot determines the most likely map, using the location estimates computed in the E-step. In conclusion, E-step corresponds to a localization step with a fixed map and M-step implements a mapping step that operates under the assumption that the robot's locations are known.

This algorithm provides two models to the robot.

- **Motion model** takes the executed action and previously estimated robot's pose as input and estimates the robot's current pose.
- **Perception model** takes a list of observations and robots pose as input and gives a likelihood of particular observation to be present.
- **Feature-based mapping** is another common approach in map building and it is most prevalent in environments that are small and have more features. As indicated by Renken [2], Feature-based methods localize features in the environment. The robot is localized by using those features, i.e. known landmarks.

Harris's 3D vision system DROID used the visual motion of image features for the 3D map to determine the camera motion by tracking features using Kalman filter. The key steps in this approach are:

- **Feature Extraction**: This is the first step in any feature-based mapping method. Common algorithms used are Harris, SUSAN, SIFT, and SURF. As mentioned in [1] SIFT is the most popular feature detector used in computer vision and robotics. It recovers the features with the position, orientation, and scale. These algorithms use Gaussian method to detect the features. To match features without any error, Random Sample Consensus (RANSAC) which is an iterative method is used to estimate parameters accurately.

- **Map building**: The next step in mapping is to register the obtained data (i.e. features) and merge it to create a map of the environment. The movement of the robot between two captured frames is necessary to merge landmarks together. For this, robot odometry data is used. The least-squares procedure is used to get more accurate camera motion after a feature is matched [6]. A minimization method is deployed to reduce the error between the estimated feature and its matching observed feature. Iterative Closest Point (ICP) is the most commonly used algorithm for aligning the data.

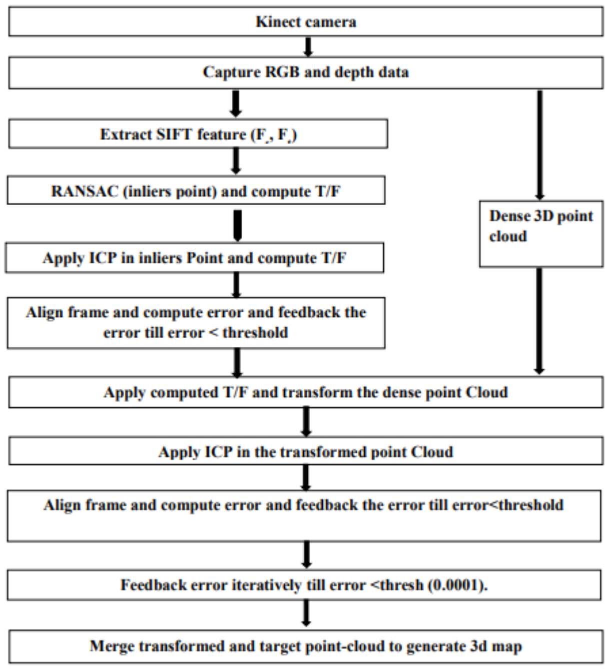

Kinect camera
Capture RGB and depth data
Extract SIFT feature (F, F)
RANSAC (inliers point) and compute T/F
Apply ICP in inliers Point and compute T/F
Align frame and compute error and feedback the error till error < threshold
Dense 3D point cloud
Apply computed T/F and transform the dense point Cloud
Apply ICP in the transformed point Cloud
Align frame and compute error and feedback the error till error<threshold
Feedback error iteratively till error <thresh (0.0001).
Merge transformed and target point-cloud to generate 3d map

Figure 2.1 Block diagram [6]

- **Map representation**: Environment maps are classified into two types of representations:

  1. **Grid-based map**

     These are accurate metric maps where a two-dimensional space is considered and the objects are placed in it with precise coordinates. As proposed by the authors in [7], it is a representation of the environment using evenly-spaced grids. These are easy to construct and maintain especially large-scale environments. Each cell in the grid (x,y) has a value which determines the occupancy of that cell. These types of maps are easy to build, represent and maintain. Grid-based maps remain a challenge due to its fine resolution, memory consumption and time taken to build a map. To build such maps, the data from the sensors must be converted into occupancy value for each grid cell, i.e. 1 if the cell is occupied and 0 if the cell is unoccupied.

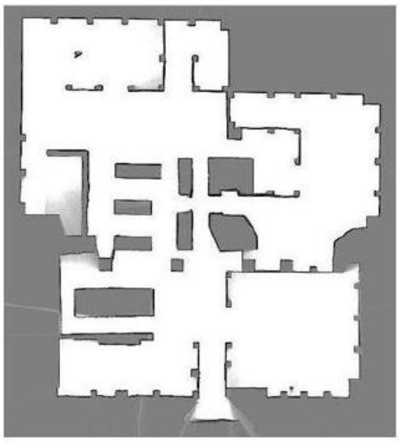
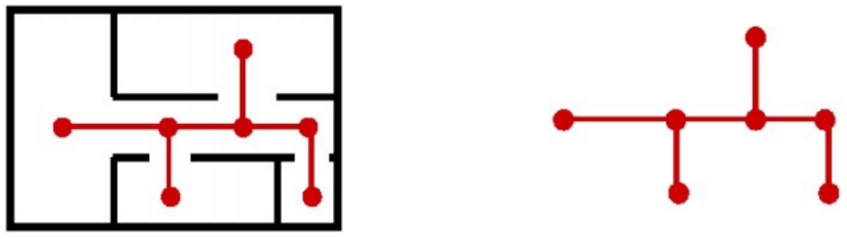


Figure 2.2 Grid-based Map [7]

2. **Topological maps**

As mentioned by [8], topological framework represent robot environments by graphs. These types of maps are used for efficient planning. The creation of these types of maps is simple and less complicated. But these maps are not as accurate as Metric maps. These do not require an accurate pose of the robot. Nodes represent gateways, landmarks or goals. Edges represent the path where the robot can navigate. Topological graphs are often extracted from a complex and detailed metric map. Kolling and Carpin [9] suggested a method to extract graph representations from occupancy grids based on the Generalized Voronoi [10]. Another common method for extracting the graph is the visibility graph [11]. Vertices are placed near the boundaries of obstacles in the environment and straight lines are used to join those vertices.



(a) (b)

Figure 2.3 Extraction from a metric graph (a), Topological graph (b)

2.1.2 Localization

To successfully navigate, the robot must obtain its pose in the surrounding environment. After getting its pose, the robot can plan a path to the goal. Generally, localization can be performed with or without a map. Mobile robots typically localize themselves in the environment using their exteroceptive sensors and environment map.

In case such a model is not available, the robot has to explore the environment on its own which is called autonomous exploration [1]. Monte-Carlo localization (MCL) is one of the current state-of-the-art approaches.

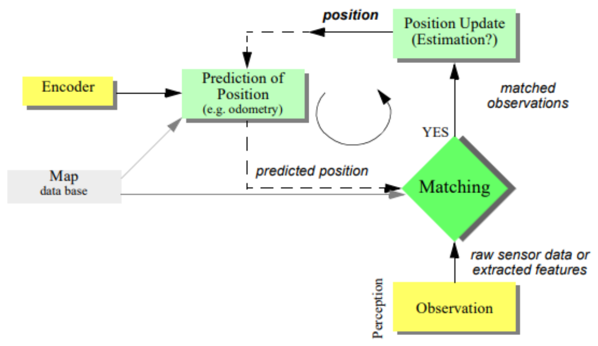

position
Position Update (Estimation?)
Encoder
Prediction of Position (e.g. odometry)
matched observations
YES
Map data base
predicted position
Matching
raw sensor data or extracted features
Perception
Observation

Figure 2.4 Schematic of Localization [1]

Probabilistic techniques: a robot must keep track of its motion by using odometry and making observations of the environment. The combination of these exteroceptive and proprioceptive sensors data enables the robot to localize itself in a given environment map (topological or metric). To get the exact position of the robot a two-step process is followed in any algorithm.

- **Action update:** Given the belief of a prior state and encoder measurements of the current state, the belief state can be modified. The belief state represents the probability of the robot being in that particular position.
- **Perception update:** This refines furthermore the robot's belief state using data from exteroceptive sensors.

2.1.2.1 Markov Localization

Markov localization can localize a robot starting from an unknown position. This algorithm initiates the state of uncertainty of the robot by a uniform distribution overall positions in the given map. Certain probabilities are assigned to each achievable robot pose in the map which is termed as belief state of the robot [12]. This assignment requires a discrete representation of a map (metric representation). To update the belief state, new data (e.g. current encoder reading) is integrated with a prior belief state. This, however, requires high computational power. Bayes rule is used to get a new belief state as a function of its sensory inputs and former belief states [12].

2.1.2.2 Kalman Filter Localization

Unlike Markov localization which uses discrete probability assignments for robots possible poses, Kalman filter-based localization proposes a well defined unique Gaussian probability density function to depict the robot’s belief state. The parameters of Gaussian distribution are updated with respect to time. This algorithm needs an initial position of the robot to localize it in a map. The robot collects data from the sensors and extracts appropriate features. The estimated features of current state from Action update are matched with the extracted features. As a last step, this algorithm uses the information from all these matches into one single unimodal Gaussian distribution and updates the robot's belief state.

$p(x)$
Markov Localization
k
$p(x)$
Kalman Filter Localization
x

(a) (b)

Figure 2.5 Discretized probability distribution (a), Continuous single hypothesis probability distribution (b) [13]

2.1.2.3 Particle Filter

Monte Carlo based technique uses multiple samples (particles) to approximate the target distribution. The particle represents the possible state of where a robot is. Like Markov's localization, this will start with a uniform random distribution of particles over the target space. This algorithm is based on Markov property that the current state’s probability distribution depends only on the previous state and the current sensor input of the environment. Bayersian Estimation algorithm is used to resample the particles recursively. After a limited number of such iterations, the particles are meant to be converging towards the actual position of the robot.

2.1.2.4 **Landmark-Based Localization**

This is a feature-based technique where feature matching is used. According to Ishiguro and Tsuji [14], the incoming images are compared with the images that are previously captured during the exploration stage to obtain the robot pose. When the sensors detect a landmark, the robot is localized using action and perception update. As the robot moves to no-landmark zone, only action update occurs and the uncertainty builds till it senses another landmark. Thus the efficiency of this method is proportional to the accuracy of the odometry values. This method is less commonly used in indoor localization, as there can be a lot of similar features in an indoor environment.

2.1.2.5 Beacon Based Localization

This is the most reliable localization when the target environment where the robot moves is fixed. This is often used in industrial and military environments. Active beacons are designed and placed in the environment. GPS system can be considered as an example of this kind of localization. The robot must know the positions of the beacons in prior. When a robot is placed in such an environment, the robot can identify its position using geometric principles.

2.1.3 Path Planning

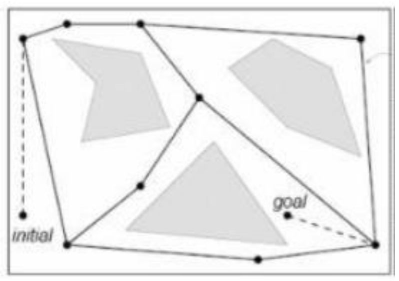
Once the current pose of the robot and the goal is known, the robot should plan a trajectory to the goal. The robot should successfully reach the goal while avoiding all obstacles. Two important terms must be defined here, these are world and configuration space. World space is defined as the environment space which embeds the robot. Configuration space can be defined as a vector containing the pose of the vehicle.

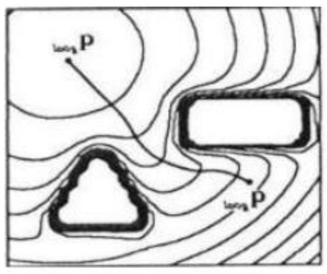
2.1.3.1 **A* Algorithm**

This is one of the widely used path planning algorithms in mobile robotics. The whole configuration space is discretized as nodes. The start point and endpoint are given to this algorithm. From the given starting node, this algorithm tries to find a path to the specified goal node while connecting to other nodes that have the lowest weights. The weight of a node is determined by two parameters. One, the heuristic distance (Manhattan, Euclidean or Chebyshev) [15] of the start to the goal node and two, the length of the path from the initial state. A tree of paths are originated at the starting node and they are extended one node at a time until the tree reaches its destination, i.e. goal node). This algorithm can explore huge areas of the map.

2.1.3.2 Probabilistic Roadmap

Here random samples are taken in the configuration space (i.e. environment map) of the robot and these samples are marked whether they correspond to free space or an occupied space (i.e. obstacle). Then using a local planner, the determined free space samples are connected to other nearby configurations. This process is continued until the road map is dense enough. Then these configurations and connections are added to the graph. In the next phase, which is called the query phase, a start and endpoints are added to the graph. A graph search algorithm is applied to the inputs and a path is obtained between the starting and goal configurations. This is done by a Dijkstra's shortest path [16] query.





(a) (b)

Figure 2.6 Probabilistic road map (a), Potential field path (b)

2.1.3.3 **Potential Field Approach**

Any physical field that adheres to Laplace's equation is a potential field. Electrical, magnetic and gravitational fields are some examples. This approach generates an artificial potential field against world space. This algorithm generates two types of fields. The attractive field where a goal node is given to this field and a Repulsive field where obstacles and boundaries are given to this field. The starting node is assigned with high potential. Thus, the robot moves from highest to lowest potential [17].

The above algorithms generate two paradigms of plans. These are Global and local path plans. The global plan is a path generated between start and goal points on the map. Whereas a Local plan is generated when the robot actually starts its motion. The local planner algorithm follows the global plan and gets modified based on the dynamic changes in the environment. This helps in avoiding dynamic obstacles (which are absent at mapping and localization phase). Some of the local path planning approaches are the global dynamic window approach, the curvature velocity method, and the Schlagle approach.

2.1.4 SLAM

When a robot is stationed in an uncharted environment, SLAM builds a consistent map incrementally while simultaneously determining its location within the map. A successful map is obtained when the pose of the robot is known without any error and the robot's pose is estimated when the map of the environment is known. SLAM is a combination of mapping and localization tasks. The interdependency between mapping and localization makes SLAM a challenging task. There are several uncertainties in mapping and localization models. So, a probabilistic approach is well suited for solving the SLAM problem. There are two steps in this approach, motion and observation model. The probabilistic motion model estimates the new robot's pose from the prior estimate and the current control law. The observation model gives the probability of making an observation taking environmental map and robot pose as inputs. The mapping and localization tasks have a lot of uncertainties in computation. So, a probabilistic approach is best suited in solving the SLAM problem. A general description of the SLAM problem is represented in figure 2-7. The current state of the robot is represented by xt, and the map (landmarks) are predicted using all current and previous observations and control law. It is described by P(xt,m|z1:t,u1:t).

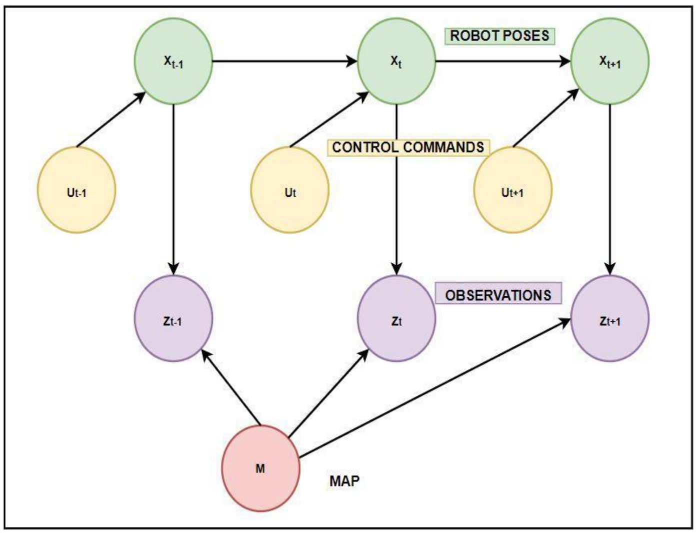


Figure 2.7 Schematic model of SLAM [18]

There are two commonly followed approaches in solving SLAM:

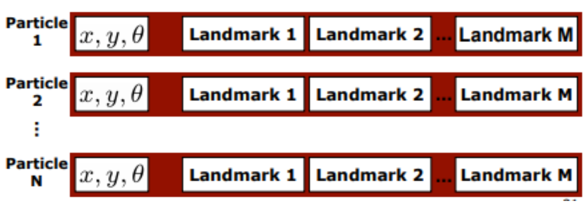
1. **Extended Kalman Filter (EKF):** This technique was introduced mathematically by Cheeseman and Smith (1986), and implemented by Moutarlier and Chatila (1989) [19]. The posterior probability distribution over robot pose including the positions of the landmarks is estimated incrementally using EKF. This is done in two steps. The first step is prediction step where EKF estimates the current pose of the robot from the current control value and previous pose estimates. The second step is the correction step where an expected observation is computed from the position estimate in the prediction step. Simultaneously, actual observation is made with the sensors. The error between both observations is used to update the estimates of the robot pose and landmarks.

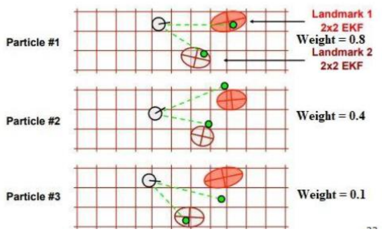
   EKF based approaches are computationally complex. Complexity is quadratically proportional with number of features. EKF algorithm uses Gaussian assumption and requires sufficiently distinct landmarks to get accurate pose update [20].

2. **Particle filter:** A particle refers to a possible state of the robot. In other words, it represents the probability distribution of the robot's pose at the current time step. Monte Carlo localization (MCL) is an application of particle filter used for robot pose estimation. Steps involved in a particle filter approach are:
   a) **Transition probability:** The probability which describes the distribution of the next state given the previous state represented by P(Xt|Xt-1). First, the transition probability is computed and a new set of particles is generated. These particles simulate the transition to the next time step.
   b) **Observation probability:** This describes the probability distribution of the observation given the location on the map. It is represented by P(Yt|Xt). Based on the observations, the particles are given a certain weight. For example, the particles which are close to the matched observation receive large weight than the particles far from the observation. In this manner, all the particles are reweighted. Resampling is done which eliminates all unmatched particles (i.e. having less weight). This process continues in a loop.

2.1.4.1 FastSLAM

It is a combination of EKF and particle filter methods. It estimates the path posterior (i.e. pose of the robot given control value and map) using a particle filter method. The beliefs of the landmarks are estimated based on the EKF filter. For each particle, there exist its own local landmark estimates (Kalman filter), since landmark estimates are conditioned on the path estimate.





(a) (b)

Figure 2.8 Particle filter landmarks (a), Weight distribution (b)

2.1.4.2 Graph SLAM

The key disadvantage of the above technique is that the data cannot be stored and offline map building is impossible. Graph Slam is state-of-the-art offline method to postpone the mapping process until the end. This represents the SLAM posterior into a graphical network containing log-likelihood of the data. There are two components motion and measurement arcs. Any two consecutive robot poses are linked with motion arcs and the feature poses are linked with measurement arcs. A nonlinear constraint represents each edge of the environment in the graph. Graph slam linearizes the set of constraints to compute a map posterior. Minimizing the sum of these constraints yields the most likely map and robot path [19].

2.1.4.3 **ORB-SLAM**

It is a Visual-SLAM used in real-time applications and can be used in any type of environment which mainly emphasis on detecting features of the environment [21].

This is an optimized SLAM solution for Stereo, Monocular and RGB-D cameras. Where the trajectory of the camera is estimated while reconstructing the environment. Here the same features are used for all the tasks: tracking, mapping, localization, and loop closure.

1. **Tracking:** Here the camera will be localized globally with every frame. First, an initial estimate of camera pose is obtained by feature matching of the current frame with the previous frame. Bundle Adjustment [18] is a process used to optimize the pose. The pose can be obtained by global localization even if the tracking is lost due to fast movements. The current frame is queried in the database containing keyframes and localization is performed using ORB feature matching and RANSAC algorithm. A visible map is extracted from the visibility graph of keyframes and matched with the local map points. This further optimizes the pose of the camera.
2. **Local Mapping:** given the camera pose and the new keyframes, a local Bundle adjustment is performed and an optimal reconstruction of the environment surrounding camera pose is achieved. New Map points are obtained by triangulating ORB features from the connected keyframes. The reconstruction of environment is done combining all the keyframes and map points.
3. **Loop closing:** The last keyframe is considered in this step. The system has embedded bags of words which is a representation of an image as a set of features. DBoW2 algorithm is used to detect the loop. The last keyframe is queried in the database to detect a loop closure. Then all the keyframes and map points are optimized and unnecessary keyframes are discarded from the local map.

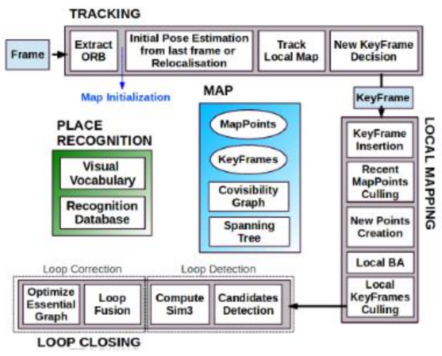


Figure 2.9 ORB SLAM [21]

2.2 ROS

The Robot Operating System (ROS) is a framework used for robot software development. The architecture of ROS is mainly composed of different components such as nodes, topics, services, etc.

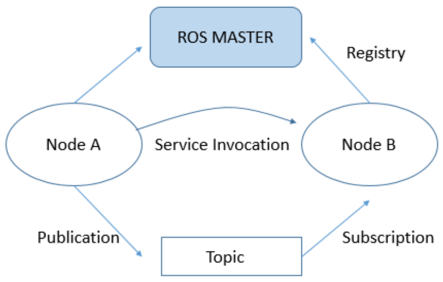


Figure 2.10 Example

Nodes are basic elements of ROS programming which perform actual functions for the robot. Each node in ROS has a name and is registered in ROS Master which keeps track of every running node. Nodes provide us a way to divide complex programs into simple ones. For example, one node controls the robot arm motor while the other is responsible for image processing.

Messages are the basis of communication in ROS. A message is a simple data structure that stores information of type integer, floating-point, Boolean or can be a user-defined data type.

Topics act like a pipeline between any two nodes to send and receive messages.

Publisher & subscriber are messaging patterns between nodes in ROS. A publisher node publishes messages into topics while subscriber node receives messages by subscribing to a specific topic. A topic is allowed to have multiple publishers and subscribers and a node can publish or subscribe to multiple topics.

Services are another type of communication architecture followed by ROS. This is a request-reply mechanism. A service is defined in a node and can be initialized through a command. A client can send a request message and service replies to that request.

2.2.1 Robot Description

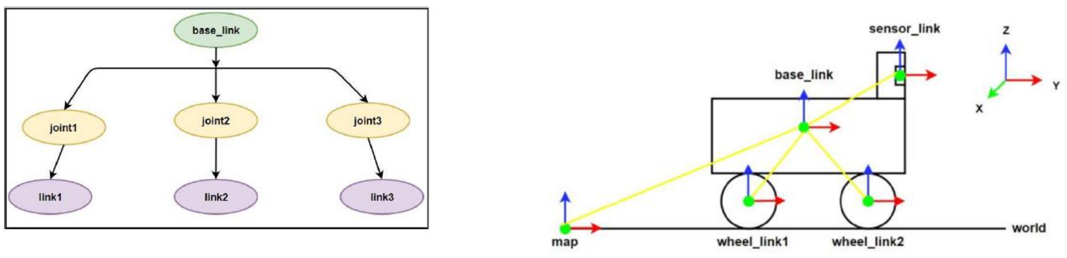


Figure 2.11 Representation of links and joints

A real robot is discretized into robot links, joints and coordinate frames. In ROS, a real robot is described in a form of URDF Unified Robot Description Format model. URDF model is a replication of the real robot and is perfect for visualization at the time of mapping and navigation tasks. *Robot_model* package is used to reconstruct the robot in ROS using the parameters defined in URDF files. *Urdf* package contains the information of robots links, joints, and sensors in the XML file format. *Joint_state_publisher* is the package which publishes the joint states in the form of sensor_msgs/JointState. *Tf* is the package used to keep track of the robot's coordinate frames and the relationship between them. It uses a transformation tree structure to keep track and update the structure with a given frequency, i.e. frequency of published transforms. *Robot_state_publisher* takes the robot model (URDF) as input and publishes all the coordinate frames and transformation between the links of the robot.

2.2.2 SLAM Packages

Real-time Appearance-Based Mapping (RTAB-Map) is the SLAM package used in this project. This is a combination of feature-based and graph-based Slam approaches.

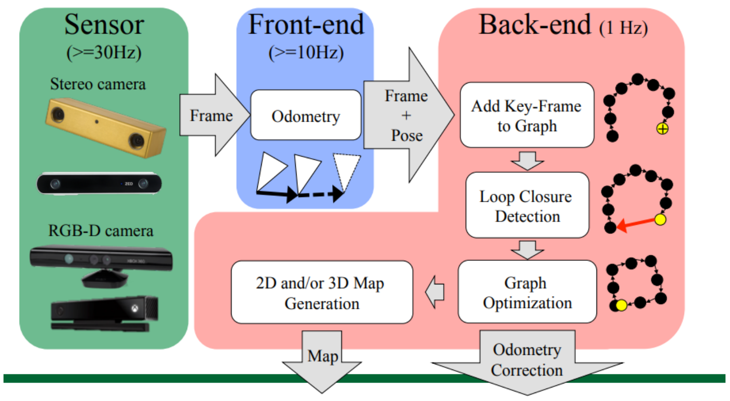


Figure 2.12 RTAB-Map Overview [22]

The high-level architecture of RTABMap contains four parts: Sensors, Front-end, Back-end and Memory management.

Sensor: the data published from the sensors is synchronized at the beginning since all the sensors need not be published at the same rate. Good synchronization is an important step to avoid errors while registering the data. The data is synchronized using the timestamps of the messages. *Rtabmap ros/rgb_sync* ROS nodelet is used for synchronizing the camera topics.

Front end (visual odometry): main tasks are feature extraction, feature matching and obtain pose for a keyframe through visual odometry. This publishes the pose of the robot in form of Odometry message with corresponding tf's transform, i.e. /odom to /base_link. Visual odometry is considered more accurate in estimating the pose than proprioceptive odometry. RTAB-Map performs this in two approaches namely Frame-To-Map (F2M) and Frame-To-Frame (F2F). First, all the features are detected in the given frame (captured by the sensors) using GoodFeatures To Track. First Feature matching is done and correspondences are registered on the Key Frame. Then, a motion model estimates the position of keyframe on the current frame using previous motion transformation. An exact transformation of the current frame relative to the keyframe is extracted using PnP RANSAC implementation of OpenCV [23]. This transformation is optimized using local bundle adjustment and the odometry is obtained. F2F analyzes current frames against the last keyframes and F2M compares the new frames with a local map of features which are created from past keyframes [24]. So, the output of the front-end module will be the frame with camera pose.

Memory management runs on top of Front-end and back-end processes in parallel. There are two types of memories prevalent in RTAB-Map. Memory management is used to optimize the working memory and long-term memory. STM is a part of working memory. It gets the pose and keyframes from Front-end module. This creates nodes that contain Odometer pose, Visual words, and a local occupancy grid. It weighs every node based on the last node in the graph. This weighing mechanism gives weight based on the importance of the locations. For example, if a location is being observed for a long time, the corresponding node gets more weight. When certain time is reached (e.g. memory threshold “Rtabmap/MemoryThr”), the nodes having smallest weights are transferred to LTM. LTM acts like a database. When a loop closure is detected in the working memory, the neighboring nodes of that location are queried and brought back to WM from LTM.

Backend: loop closure detection, graph optimization, and map generation. This module contains links which are a rigid transformation between nodes that have been created by STM. Loop closure link and proximity link are added when a loop closure is detected between two nodes. When a loop closure is found, the graph optimizer refines any errors in the pose and updates the odometry. All the links are used as building blocks for graph optimization GTSAM which RTAB-Map uses by default. After adding all links and optimizing the graph, this module generates OctoMap, 2D Occupancy Grid and Point cloud outputs for external modules (e.g. path planning packages).

2.2.3 Path Planning Packages

Move_base is the package used for path planning from one point to another. This package takes the robot tf (tf/tfMessage), environment map(nav_msgs/GetMap), pose of the robot (nav_msgs/Odometry) and the goal point (geometry_msgs/PoseStamped) as input. And gives velocity commands (geometry_msgs/Twist) to the robot base as output. This also takes sensor information from the sensors to perceive the environment in real-time. This package embeds a global and local planner for navigation. The global planner creates a trajectory from start point to the goal position based on the map (i.e. occupancy grid) given by RTAB-Map which considers only static obstacles. The local planner takes real-time data from the sensors, modifies the global path avoiding dynamic obstacles following the global planner. This package also provides costmaps. Each grid of a map is given a cost in range of 0-254 based on parameters. Cost indicates if particular position in the map is vacant or occupied.

Nav_core package provides Global planner, local planner, and recovery behavior which are plugged into move_base package through nav_core interfaces. Planners are very important for navigation. *Navfn* is a navigation function used to create plans for a mobile base. This planner uses costmap to find minimum cost plan from start point to the goal point. This assumes robot to be circular in the given occupancy grid. The obtained path is optimized using Dijkstra’s Algorithm.

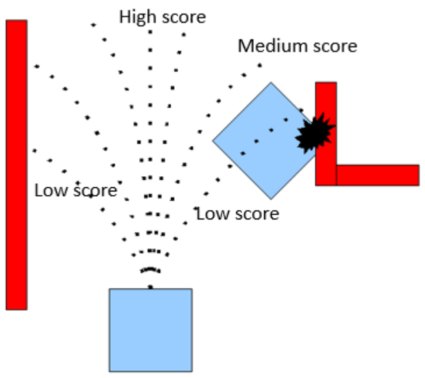


Figure 2.13 Dwa_local_planner [25]

Dynamic window approach (DWA) is followed in this project as a local planner. In DWA planner, the robot control space is sampled, i.e. velocity. For each sampled velocity, a kinematic trajectory is simulated from the robot’s current position. A score is given to each simulation based on factors like distance to the goal, distance between the path and obstacle, speed. The trajectory which has maximum score is selected as the local plan.

2.2.4 **ROS Navigation Stack**

The combination of SLAM and path planning for a mobile robot is termed as navigation stack. This is mainly used for differential drive robots. RTAB-Map gives the occupancy grid map and localization position through topics roadmap/grid_map and rtabmap/localizationpose respectively. The sensor messages are published through sensor topics rtabmap/rgbd_data and odometry is published to rtabmap/odom. These inputs are given to the move_base package where a plan is generated by a global planner.

3 TOM3D-Robot System Description

3.1 TOM3D-Robot

The main goal of this project is to enhance the autonomous navigation system of the TOM3D-Robot by developing a Visual-SLAM based navigation system using Kinect camera. To achieve this task, the robot must perceive its environment, perform perfect cognition to avoid static and dynamic obstacles in 3D and respond to sudden changes in trajectory. TOM3D-Robot is a differential drive mobile robot developed by the chair of automatic control engineering, University of Siegen. It consists of three wheels, more specifically two motor-driven wheels in front and one caster wheel in the back which supports the robot in translation and rotation. The TOM3D-Robot has two control levels namely high-level control (PC) and lower-level control (Arduino Mega 2560).

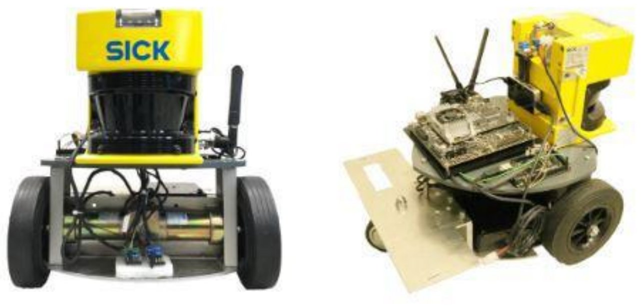


Figure 3.1 TOM3d robot

The low-level controller is a bridge between Higher-level controller (PC) and proprioceptive sensors. The main tasks of this controller are:

1. It receives ROS Messages from Higher-level control, i.e. Navigation stack and applies high-level commands to the motors.
2. It receives sensor readings, i.e. IMU and encoders, and sends the feedback to the High-level controller, i.e. Navigation stack.

The Arduino board contains multiple digital input and output pins, four serial ports (UART), six external interrupts and a 16 MHz crystal oscillator. The commands from the high-level controller are sent to the Arduino via rosserial package.

The high-level control is a computer that is connected to the robot to handle high computational tasks. Since the Arduino controller can't process exteroceptive sensor data because of their high computational needs, PC is used. The PC has a Linux operating system with ROS framework and entire Navigation stack packages installed.

The robot navigation system has two operation modes namely manual and autonomous modes. The manual mode is used to drive the robot manually creating a map of the environment and localizing itself in that map. The robot can be moved manually with the help of a wireless Logitech gamepad. The joystick is connected to the high-level controller using a nano wireless USB adaptor. ROS drivers convert raw commands of the gamepad into ROS velocity commands which then are sent to Arduino. The Arduino receives ROS commands through the topic "/cmd_vel" and controls the motors accordingly.

After the generation of the whole map and successfully localize the robot in that map, Autonomous mode is activated to navigate the robot autonomously to the desired location. Switching between the two modes is done by selecting the appropriate button on the gamepad. In the autonomous mode, control velocity for motors is given by the navigation stack packages, i.e. motion planner instead of gamepad. Here, when the goal is given to the system, the navigation stack processes all acquired information and outputs velocity commands to the Arduino board through rosserial package. Which is then applied to the motors.

3.1.1 Sensors and Data Handling

Arduino board is used for collecting and processing sensor data, i.e. wheel encoders and IMU data. Moreover, It applies necessary control actions to the motors based on the high-level controller commands. Encoders give the wheel velocities which are required to calculate the pose of the robot. When the wheel starts rotating, each encoder gives two square wave signals with 90 degrees offset as output. These two channels are connected to Arduino digital input pins. A UM7- LT orientation sensor is the IMU used in this project. The connection between IMU and Arduino is established using UART serial communication. Both encoders and IMU are key sensors in estimating the pose of the robot.

(a) (b)

Figure 3.2 (a) Arduino board, (b) IMU [26]

3.1.2 **Motor Control**

In this project, two 12 v PITTMAN motors are used to drive the wheels of the robot. These are controlled by PWM signals from Arduino. Arduino receives motion, i.e. velocity commands from high-level controller and converts them to appropriate PWM signals.

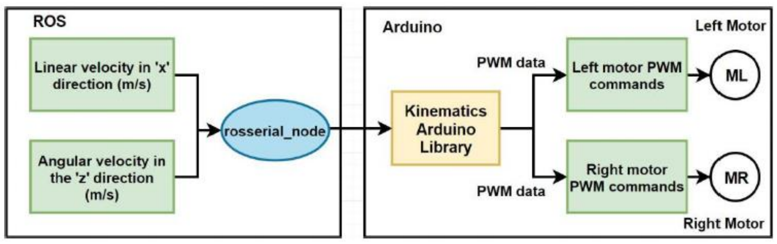


Figure 3.3 Kinematics library workflow [27]

Velocity commands are published to the ROS system either by joystick in manual mode or by navigation stack in autonomous mode. Rosserial lets Arduino subscribe to ROS system topics to receive the velocity commands. The conversion of velocity commands to PWM signals is done using a kinematics library developed by Linorobot developers [27]. The accurate movement of the robot is the basis of perfect navigation. This includes error-free application of the velocity commands to the motors. But, due to the frictional losses, there can be accumulated errors. To avoid this a PID controller is implemented. PID node takes set point velocities and measured velocities as input and gives necessary control values to follow the reference. The rosserial node publishes the motor's velocity in a message of type tom_3d_control/Wheelvelocities on two topics *leftmotor/wheel_velocities* and *rightmotor/wheel_velocities*. The output of the PID node is published to *wheel_velocities_pid* topic with message type std_msgs/Float64. This control value is given to kinematics library by rosserial to compute the required PWM signals.

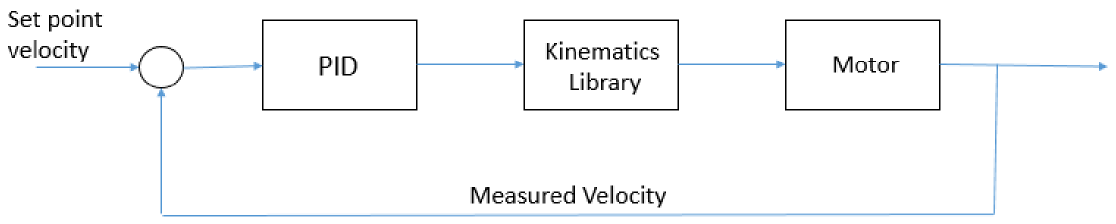


Figure 3.4 Control Loop

3.1.3 Joystick

Logitech F170 gaming controller is used in this project which has interface drivers in ROS. It is connected to an onboard computer using a Nano wireless USB receiver. The interfacing between joystick and ROS is carried out using ros-kinetic-joy package. The joy-node in that package converts joystick commands into an array of state values. A teleop node is used to translate that state values which are of form sensor_msgs::joy to geometry_msgs::twist. switching between modes is done by pressing the appropriate button in the gamepad. A package developed by irlab is used for this functionality. Barc_teleop node takes sensor_msgs::joy as input and outputs geometry_msgs::Twist to the topic /teleop/cmd_vel. Barc_safe_cmd node subscribes to above topic and /navigation/cmd_vel topic which has twist messages published by navigation stack. Based on the user’s selection this node publishes commands to /cmd_vel topic for further control of the robot.

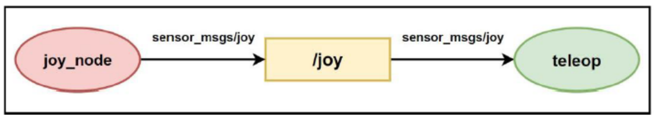


Figure 3.5 Joynode [27]

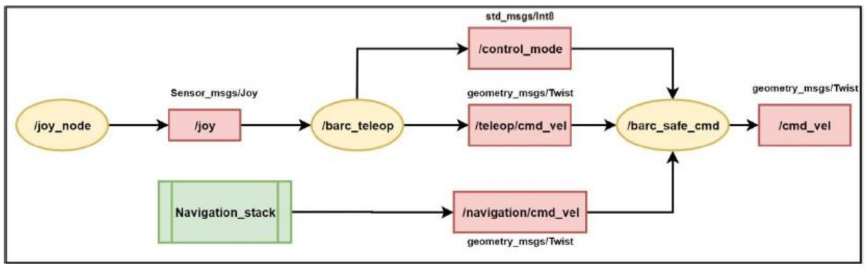


Figure 3.6 Switching modes [27]

3.2 Power Supply

Table 3-1 describes the power requirements of the whole TOM3D-Robot system. A fixed supply voltage of 25V and 3A is used to power up the system. Where DC-DC converters are used to step down the 25V to different hardware power requirements.

Table 3-1 Power requirements

Arduino board	9V, 0.5~0.6A
Encoders	5V
EC Motor	12V
Laser Scanner	24V, 0.6A
On- board computer	19.5V,4.62A
Kinect Camera	12V, 2.67 A

4 System Implementation

4.1 System Architecture

The project uses simple system architecture as described in section 3.1. The low-level controller which is Arduino microcontroller reads sensor data like IMU and wheel encoders and sends it to the high-level controller which is a PC. The joystick which is used to teleoperate the robot is connected to the PC. Kinect camera which is the main RGBD sensor used in this project is also connected to the PC. The sensor data is processed in the PC through corresponding ROS packages and velocity commands are packed in ROS message format and sent to the Arduino microcontroller. The Arduino microcontroller transforms the velocity commands in ROS message to Pulse Width Modulation (PMW) signals to control the motors.

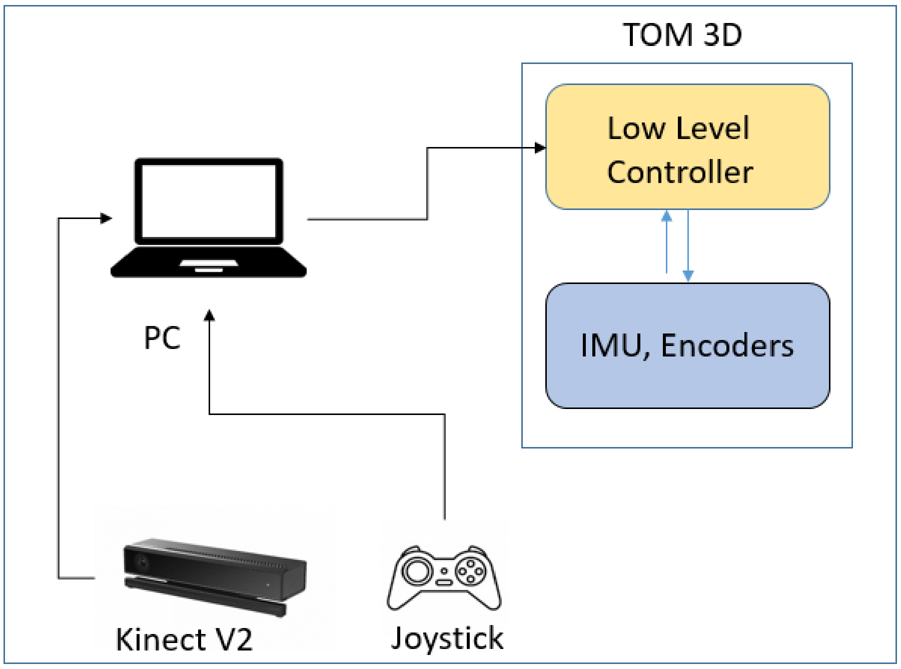


Figure 4.1 System Architecture

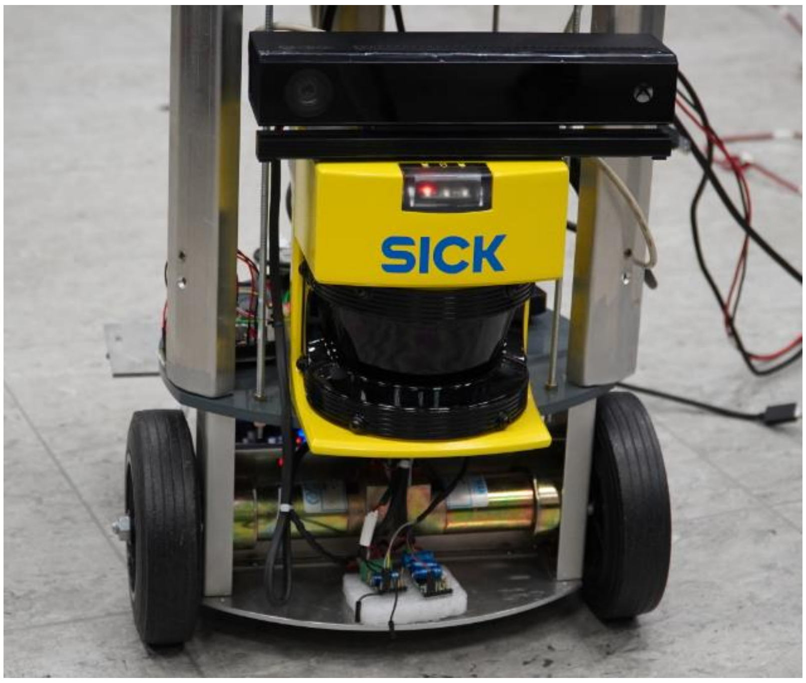


Figure 4.2 Hardware Setup

4.2 TOM3D Robot URDF Model Description

Unified Robot Description Format is an XML format for representing a robot model in ROS which is used to visualize and simulate the robot in tools like Gazebo and Rviz. First a 3D model of robot is created using Solidworks and converted into URDF XML description file using URDF exporter plugin provided by Solidworks. The accurate dimensions of the TOM3D robot are taken and modeled accordingly. The 3D model of Kinect camera is sourced from an open-source community named GrabCAD. Each individual part of a robot is modeled and configured with a definite coordinate axis, material properties and then assembled to a global coordinate axis of the robot. Based on the above-mentioned configurations, the exporter generates URDF file for the robot which also includes launch file used to bring up the robot in simulation tools like rviz and Gazebo.

Tf Frames

Robot_state_publisher node subscribes to the URDF model of the robot. The tf published by the above node broadcast the state of each link with respect to the base of the robot. A static transform is published in the launch file between camera frame and the base of the robot, i.e. between *camera_link* and *base_link*.

The main transform used in this project is the tf from the base of the robot to map frame that is loaded by the stack. To localize the robot in the given environment, the point cloud data published from the Kinect frame has to be transformed to the map frame. The transform from base frame of the robot to sensor (i.e. Kinect) is achieved by publishing a static transform. The rtabmap/rgbd_odometry node which is visual odometry provided by rtabmap package provides the transformation from base frame to the map frame.

RTAB-Map node takes RGBD input from the camera and computes Odometry information and publish it to the topic Odom (to be mapped from the original name) with a message type of nav_msgs::odometry.

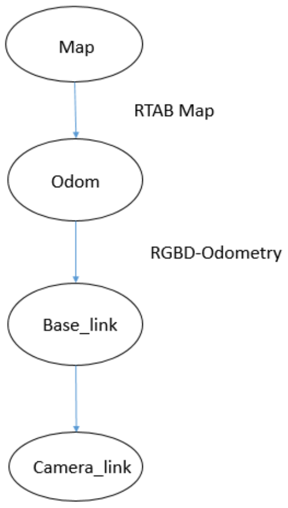


Figure 4.3 Transforms from Map to Camera

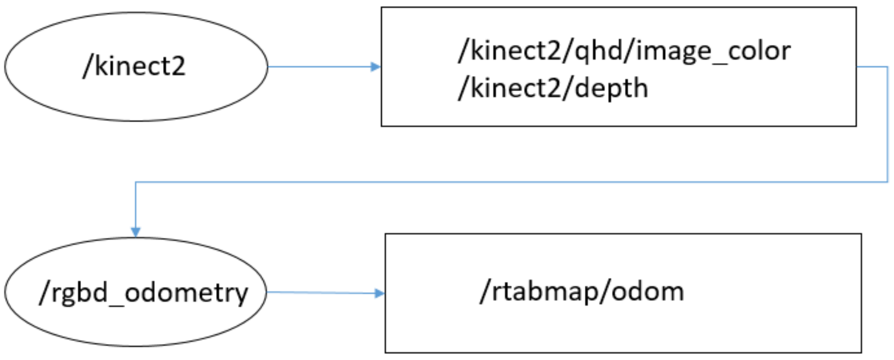
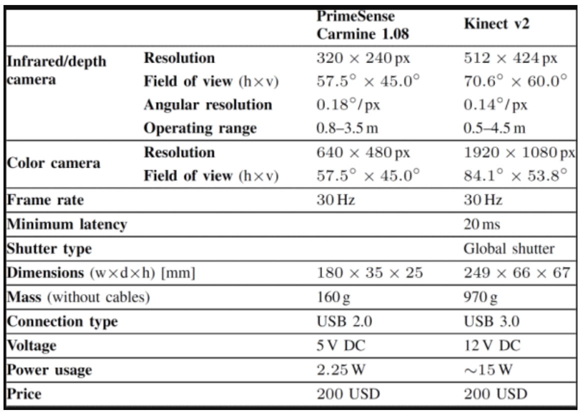


Figure 4.4 Odometry transformations

4.3 Kinect Camera

Kinect is an RGB-D camera developed by PrimeSense - an expert company in high-performance 3D machine vision technologies and licensed by Microsoft. It is used as a gaming accessory for Xbox 360 and capable of recognizing human bodies and gestures [28]. Its low-cost sensor features drive this product for research purposes in robotics and computer vision fields where robot navigation based on dense and accurate environment maps can be realized. Kinect v2 has two cameras namely color, Infrared camera, and one laser-based IR projector. The Kinect camera resolution is 1920*1080 pixels (VGA size) and has a field of view of 70 degrees horizontal and 60 degrees vertical. The rest of the specifications are listed in figure 3.2.

The camera is based on the time-of-flight (ToF) principle which offers high resolution and a wide field of view. A strobed infrared light is emitted on the environment. The light reflected from the obstacle is captured. The time of emission and reflections for each pixel are captured by the infrared camera. It uses wave modulation and phase detection to estimate the distance to obstacles.



		PrimeSense Carmine 1.08	Kinect v2
Infrared/depth camera	**Resolution**	320×240 px	512×424 px
	Field of view (h×v)	$57.5^\circ \times 45.0^\circ$	$70.6^\circ \times 60.0^\circ$
	Angular resolution	0.18°/px	0.14°/px
	Operating range	0.8–3.5 m	0.5–4.5 m
Color camera	**Resolution**	640×480 px	1920×1080 px
	Field of view (h×v)	$57.5^\circ \times 45.0^\circ$	$84.1^\circ \times 53.8^\circ$
Frame rate		30 Hz	30 Hz
Minimum latency			20 ms
Shutter type			Global shutter
Dimensions (w×d×h) [mm]		$180 \times 35 \times 25$	$249 \times 66 \times 67$
Mass (without cables)		160 g	970 g
Connection type		USB 2.0	USB 3.0
Voltage		5 V DC	12 V DC
Power usage		2.25 W	~15 W
Price		200 USD	200 USD

Figure 4.5 Specifications [28]

4.3.1 Kinect Camera Interface

The Kinect V2 camera is connected to the PC using a USB 3.0 port. To acquire the Kinect camera data, *libfreenect2* driver is used to interface Kinect V2 camera with ROS environment. This driver also extracts raw data and classifies it into RGB, IR and Depth Images which can be sent in the form of ROS message types. IAI Kinect 2 is a collection of different tools to create a user-friendly environment for utilizing data from camera in different robotic applications. It works on Ubuntu 16.04 and ROS kinetic distro.

4.3.2 Kinect Camera Calibration

Calibration is done to estimate the parameters of any sensors before the sensor is used to get data. For a camera sensor, calibration estimates the parameters like focal length and optical center and rectifies any distortion in the camera's vision. These parameters play a crucial role in modeling transformations of coordinates of external objects into their images [22]. For each and every extracted point (feature) with Homogeneous coordinates in the environment (external world frame), is transformed into coordinates in camera frame and is finally transformed into pixel coordinates.

In this project, calibration is done using *kinect2_calibration* package which is a part of IAI Kinect 2 package set. Calibration is done by either chessboard of circular or square patterns. The pattern is printed and glued to a flat surface. The chessboard is held in front of the Kinect camera and recording program is started to record images (RGB and IR) with different orientations of the pattern. This tool collects RGB color images from the RGB sensor using the installed driver. Using the data recorded by the RGB camera, the intrinsic parameters like focal length, the optical center and the skew coefficient of the camera are calibrated by the package. Coordinates of specific feature points of the calibration object (checkerboard) are known. This node extracts coordinates of feature points from individual images and then finds out optimum transformation of coordinates for all feature points in all views. This calibration is done to both RGB and IR cameras.

In the second step, images are recorded with both RGB and IR simultaneously. Using the calibration parameters obtained for RGB and IR, depth measurements are calibrated. All the calibration results are stored in a directory in .yaml file format. Kinect2 Bridge is a package that connects libfreenect2 driver and ROS. It reads the calibration parameters and converts sensor input to ROS topics. The topics published by this package are mainly classified into HD Topics (Full HD resolution 1920x1080), Quarter HD (960x540) and IR/Depth Topics.

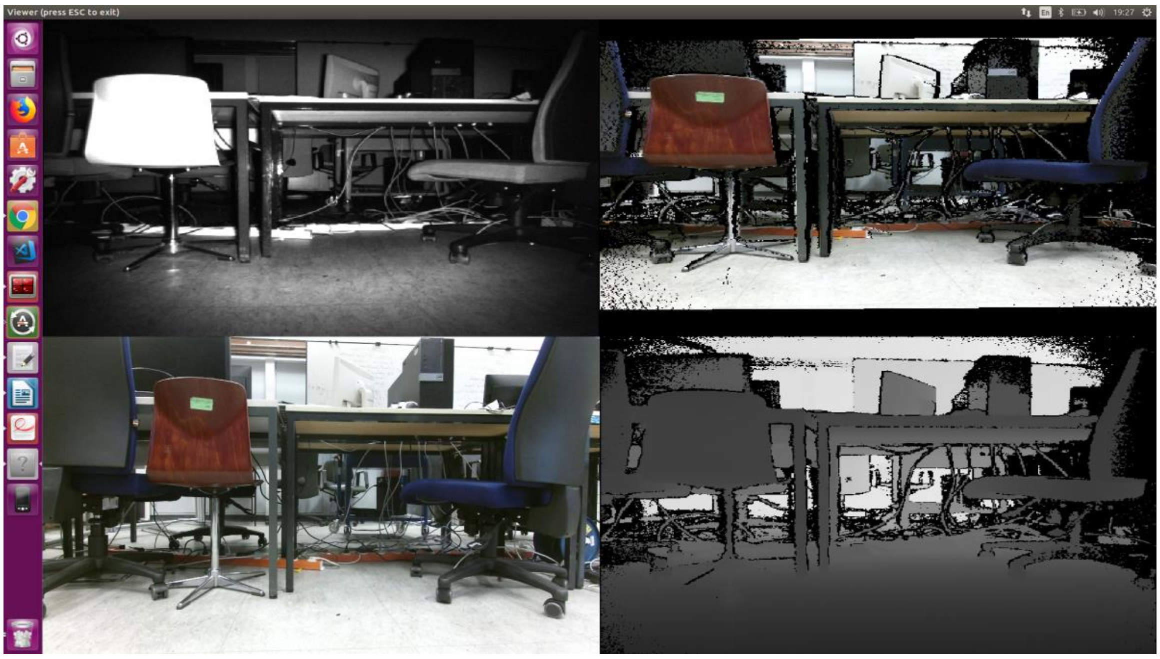


Figure 4.6 Kinect V2 output picture format

4.4 Overall System setup

The point cloud data from the Kinect camera is used to build a map and localize the robot in that map. In autonomous navigation the map is given as input to the navigation stack, see section 2.2.4. As discussed in section 4.1.1, kinect2_bridge node publishes the sensor data on to the topics remapped as:

rgb_topic:=/kinect2/qhd/image_color_rect;depth_topic:=/kinect2/qhd/image_depth_rect;camera_info_topic:=/kinect2/qhd/camera_info.

RTAB-Map uses the above topics as input for mapping and localization. *Move_base* is the path planning node of the navigation stack. It provides path and motion planning algorithms for autonomous navigation. After successful mapping, RTAB-Map provides the grid-based occupancy map -as discussed in section 2.1.1- of the environment to navigation stack through the topic "/rtabmap/grid_map" in the form of message type nav_msgs::OccupancyGrid. After localization, RTAB-Map publishes the estimated pose of the robot through /rtabmap/localizationpose topic. Then a goal is set, which is an input to the move_base package. This publishes velocity as geometry_msgs:: twist message to the topic /cmd_vel for navigation along the planned path.

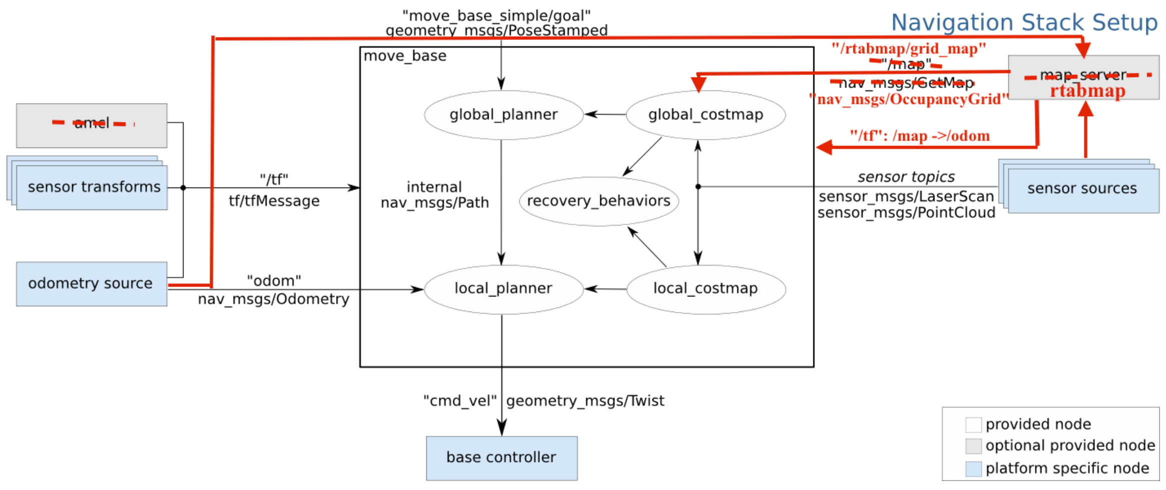


Figure 4.7 Move_base node [29]

4.4.1 RTAB-Map Node

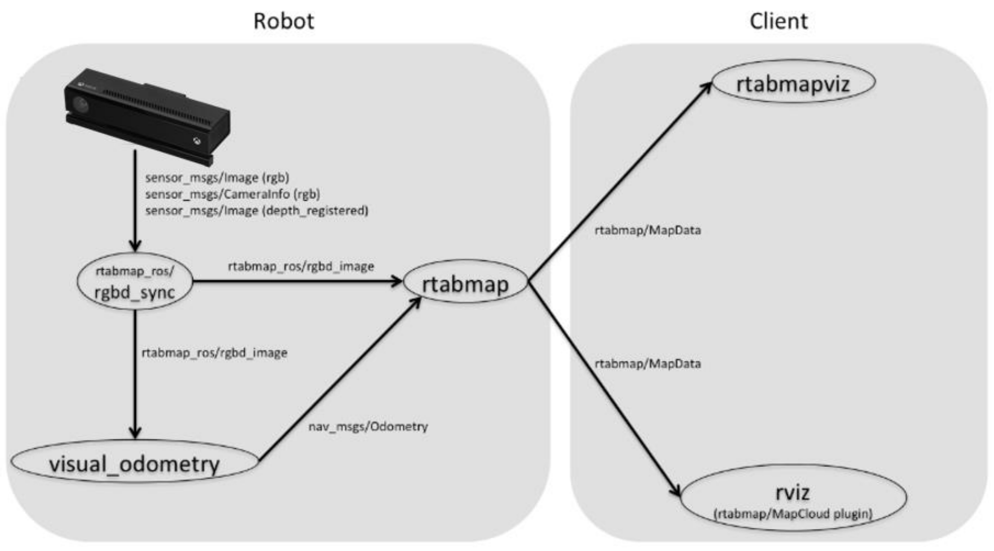


Figure 4.8 RTAB-Map skeleton [24]

As discussed in section 2.2.2 RTAB-Map is the SLAM package used in this project. The rtabmap node is the key node in performing mapping and localization tasks for the robot using a camera or any other sensors. It is a wrapper of RTAB-Map core library. Rtabmap node uses appearance based SLAM as mentioned in section 2.2.2. This algorithm uses data collected from the vision sensors (Kinect) to localize and map the environment simultaneously. The localization information comes from visual odometry node "Rtabmap/rgbd_odometry". Rgbd_odometry node receives RGB-D information from the sensor (Kinect) via rgbd_sync node which synchronizes all RGB and depth topics. The odometry data is sent through topic “rtabmap/odom “. By default, rtabmap node is set to be launched in mapping mode. All the required nodes, parameters and the information about remapped topics are in the launch file “rtabmap.launch”.

The robot is moved in the environment manually and rtabmap node is launched to start mapping. While the robot moves, the map is expanded and new images are compared with already detected feature images. The map is optimized when the algorithm finds a loop closure. Thus, in the front end, this node constructs a map interpreting sensor data and getting odometry while the robot transverses in the environment. The optimized graph is published as output from this node. It also publishes 2D Occupancy grid map which is used by the navigation packages through topics *grid_map/proj_map* (these are remapped according to the package requirements). The map is stored in the local directory “~/.ros/rtabmap.db”.

Now after successful mapping, the mode is set to localization in the visualization tool as shown in figure 4.9.

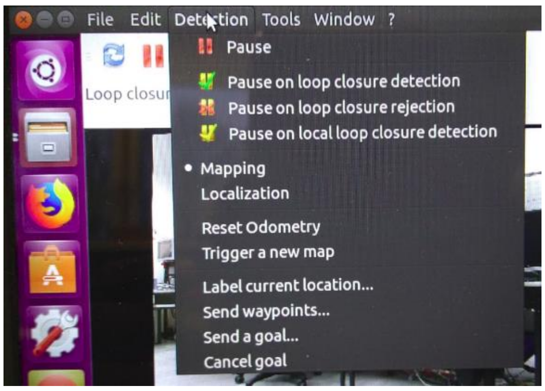


Figure 4.9 UI for changing modes

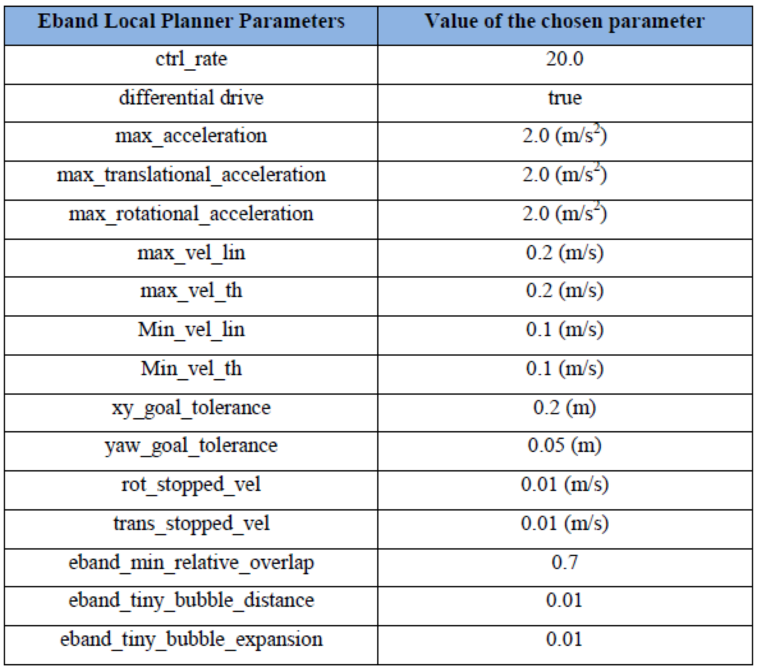
In this mode, the incremental building of map is stopped. The current frame is compared with locations in the map and RTAB-Map localize itself by finding a loop closure. When a loop closure is found, the odometry is corrected and the current cloud will be aligned to the map. So the output from the RTAB-Map are:

- Occupancy grid map in the form of “navmsgs/Occupancygrid” message type which has information (metadata) about the map database published to “/rtabmap/grid_map “ topic and
- A localization pose in the form of “geometry_msgs/PoseStamped” message type which is further used by navigation stack for autonomous navigation.

4.4.2 Global and Local Planners (move_base node):

As stated earlier move_base is used as the navigation node which embeds global and local planners. These planners help to generate a path from the start point (localized position) to the goal and plan the motion of the robot in that path accordingly. The global planner helps in generating a global path based on the static obstacles which are mapped in the grid map published by RTAB-Map node. The local planner helps in providing the motion plan for the robot by following the global path.

The global planner uses global costmap that is generated from the "/map" topic. The default global planner used in this project is the *navfn*. This considers the cost map value of every cell present in the grid and plans a global path accordingly. The local planner publishes the velocity commands to the robot base based on the global path. *Eband_local_planner* is used as a local planner in this project. To activate the local planner, the base_local_planner is set to eband_local_planner/EBandPlannerROS while launching the move_base node. The local planner parameters are tuned for the robot manually to get better results. The list of parameters used in this planner is shown in figure 4-10.



Eband Local Planner Parameters	Value of the chosen parameter
ctrl_rate	20.0
differential drive	true
max_acceleration	2.0 (m/s^2)
max_translational_acceleration	2.0 (m/s^2)
max_rotational_acceleration	2.0 (m/s^2)
max_vel_lin	0.2 (m/s)
max_vel_th	0.2 (m/s)
Min_vel_lin	0.1 (m/s)
Min_vel_th	0.1 (m/s)
xy_goal_tolerance	0.2 (m)
yaw_goal_tolerance	0.05 (m)
rot_stopped_vel	0.01 (m/s)
trans_stopped_vel	0.01 (m/s)
eband_min_relative_overlap	0.7
eband_tiny_bubble_distance	0.01
eband_tiny_bubble_expansion	0.01

Figure 4.10 Parameters [25]

The base_local_planner package acts as a controller for the robot. This package is a bridge between the path planner and the real robot. It analyzes the map and creates a kinematic trajectory for the robot with the loaded parameters. Two main tasks are done by this planner.

The first task is to get the grid values from global and local costmaps and generates a path which follows the global path. Then it gives velocity commands to the robot.

Local and Global Cost Map Parameters

The package used for local and global costmap generation and updating is costmap_2d. This takes input from "/map" sent by RTAB-Map as a grid map. It analyses static obstacle information from the grid map and assigns respective weights to the static obstacles. All this information is sent to the global planners which in turn creates a global path.

Costmap_2d takes sensor data of the environment as input and builds an occupancy grid that assigns cost to the map (local costmap). The creation of local costmap is maintained in two layers by costmap_2d node. The static layer is loaded with the map provided from RTAB-Map node. When an obstacle is sensed, Marking is done on top of the static layer and this dynamic information about obstacle is stored in obstacle layer. When an already mapped obstacle is moved from its position, a clearing operation is performed on the static layer. Marking and clearing operations are performed with a certain frequency defined in parameters and update the costmap.

Footprint: The robot's boundary is to be loaded to the costmap_2d package programmatically. This is crucial in performing an obstacle avoidance task. Robot's dimensions are loaded as an array of coordinates with respect to the loaded map frame. The exact dimensions of the robot are taken into consideration and a footprint is created around the robot. The footprint of the robot is displayed in figure 4-11.

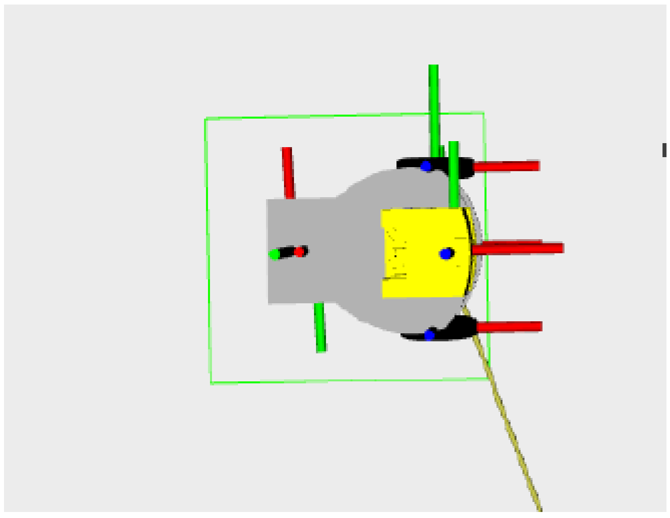


Figure 4.11 Robot Footprint

Obstacle layer parameters: Detection of obstacles is done by adding parameters in this context. These parameters are added in the form of .yaml files and are loaded in the navigation package. The observation source is an important parameter that defines the source of sensor data. Type of obstacles to be avoided are pre-estimated and the parameters maximum and minimum height are set accordingly.

This implementation failed to update the local cost map when a dynamic obstacle, i.e. obstacle which is encountered while the robot is autonomously navigating. In this case, Costmap_2d node is unable to get the point cloud data (i.e. obstacle information) from the Kinect camera topic. Thus, costmap_2d node is unable to mark the dynamic obstacles on the costmap and base_local_planner node which is responsible to create a local path is failing to avoid the dynamic obstacles.

4.4.3 Software Architecture

The software architecture of this project is divided into three levels:

1. Navigation stack layer: the navigation stack performs all the mapping, localization and path planning tasks and publishes necessary commands for the robot to navigate through *navigation/cmd_vel* topic.
2. safe_cmd_mode layer: it is responsible for switching between manual and autonomous mode.
3. The rosserial layer: it is responsible for serial communication between ROS and the robot. Rosserial takes the velocity commands from */cmd_vel* topic and converts it into velocities (rpm) which are given as a reference to the PID controller to control motors velocities.

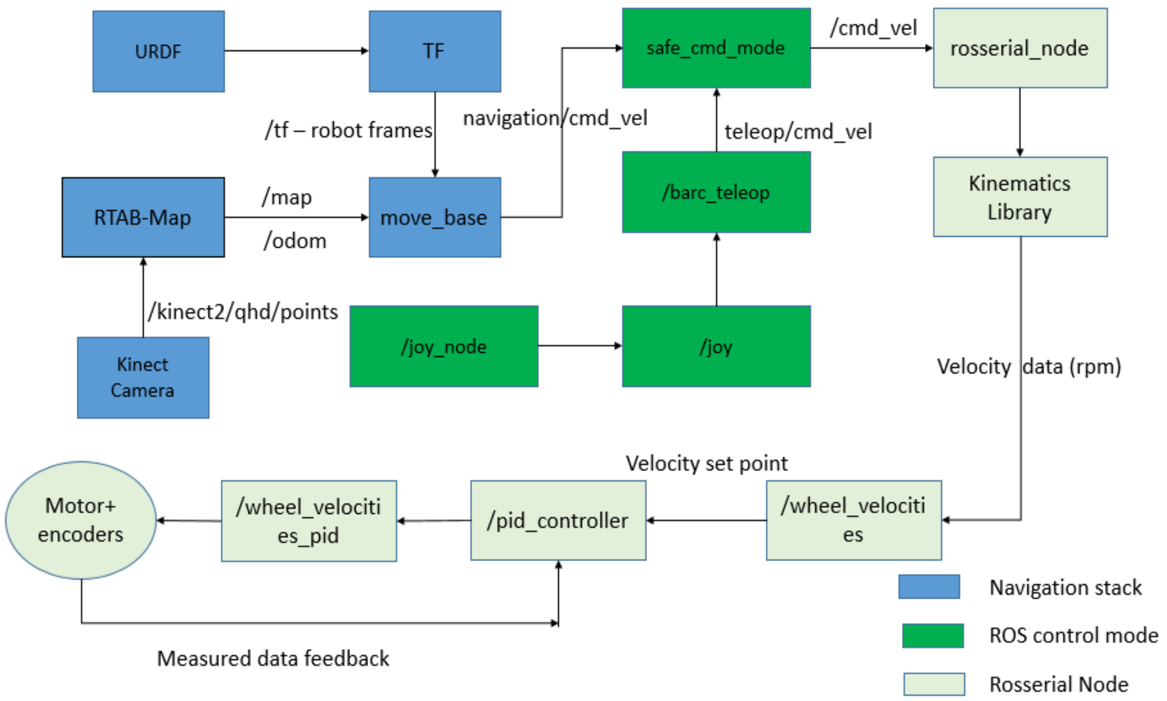


Figure 4.12 Software Architecture

5 Results

5.1 Mapping

First, the camera node is launched to get 3D point cloud data of the environment, then the robot's tf is published which is the input to rtabmap node. Rtabmap node is launched in mapping mode and the robot is driven manually using the joystick. The robot is started at a fixed point in the environment and made to move along the edges of the lab. The robot is stopped at the same starting point so that it can find a loop closure and the map is optimized. Figure 5-1 shows the map being built incrementally. The green color in the background indicates detection of the loop closure. The pink and blue paths in the map define the trajectory of the robot. Here the map is built in two sessions where each color represents a session's trajectory. The unmapped area is represented in black color where the robot cannot see the environment due to mechanical constraints and the point clouds outside the map boundary are outliers. Rtabmap gives 3D Map which contains dense point cloud data and a 2D occupancy grid where the 3D objects are projected into 2D plane. The grey color of the pixels indicates the free space in the environment and the black color indicates the obstacles. All the point cloud information and map is stored in the form of "rtabmap.db" database which can be accessed in future.

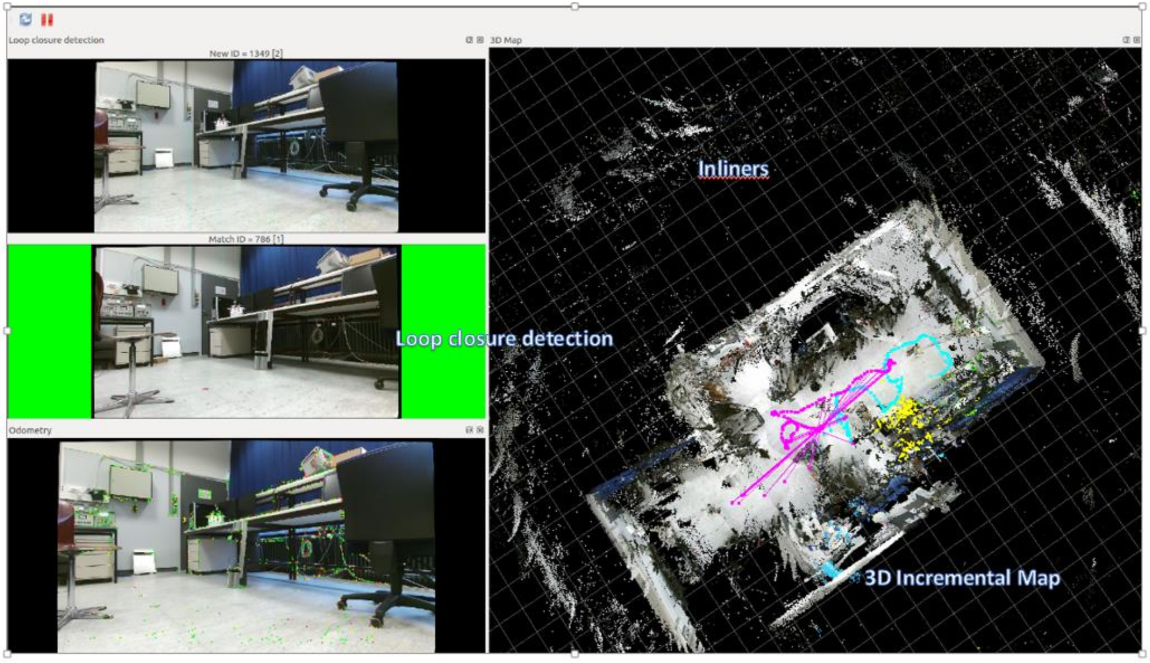


Figure 5.1 Map Generation

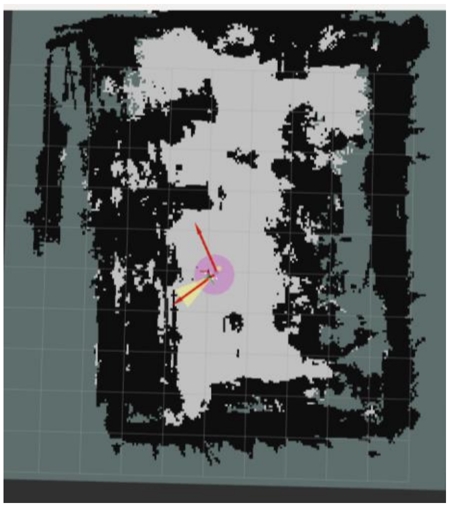
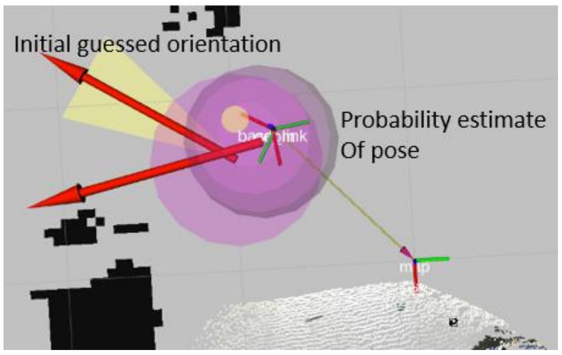
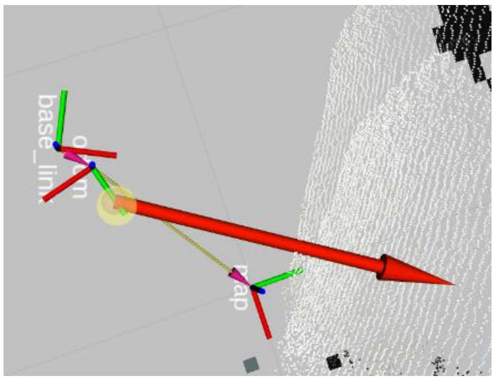


Figure 5.2 Occupancy Grid

5.2 Robot Localization

When the map is built successfully, the mode is changed to localization in the RTAB-Map. The robot is made to rotate in-place so that it detects the features and matches them with the database. First, the algorithm estimates a guess of localization pose which has an uncertainty ellipse and orientation around it. Figure 5-3-(a) displays the initial guessed orientation of the robot and the probability estimate of the pose. Now the robot is made to rotate around, then it acquires a localization pose estimate which matches the features it sees.





(a) (b)

Figure 5.3 Localization estimate (a), Localization pose after successful localization (b)

Now the map data and the localization pose are subscribed by the move base package. The goal is set in RVIZ and the navigation stack plans a path, sends velocity commands to the base_link to navigate the robot.

To check the accuracy of this localization, two tests are conducted on the robot. First, the robot is started at the origin and moved to different locations inside the lab. The estimated localization position is noted at every point and the actual position which the robot has moved is measured in the real world. The results are plotted in Figure 5-4. The maximum error in X-direction and Y-direction is 15cm and 10 cm respectively. The results are better compared to AMCL localization using LIDAR sensor.

In the second test, the robot is moved several times to a random position inside the lab, and the estimated and actual position of the robot is noted each time in the lab. The results are plotted in Figure 5-5.

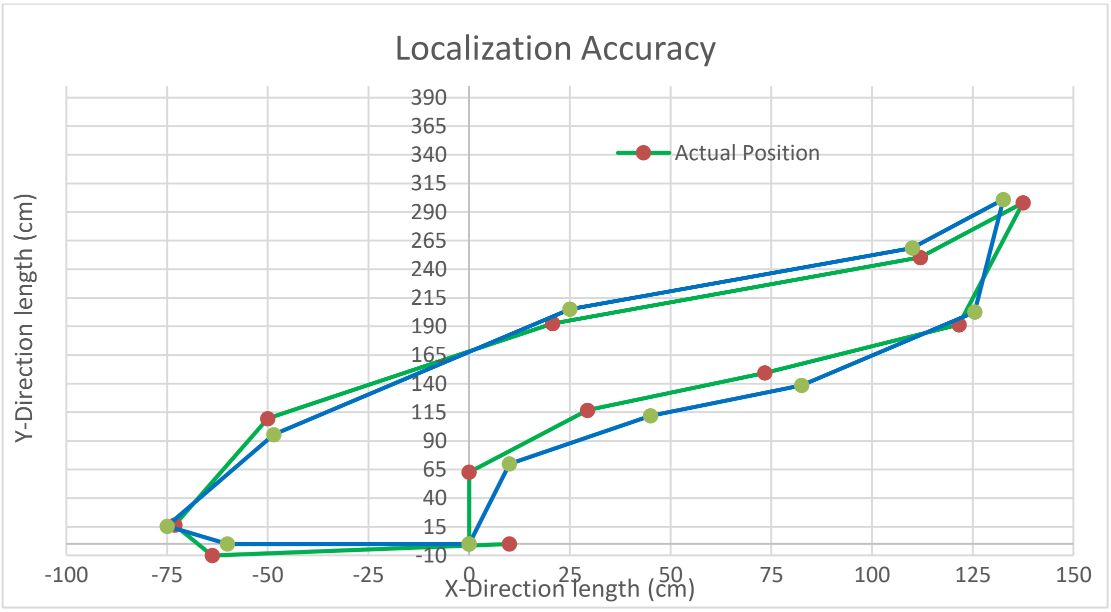


Figure 5.4 Actual position and RTAB-Map estimation

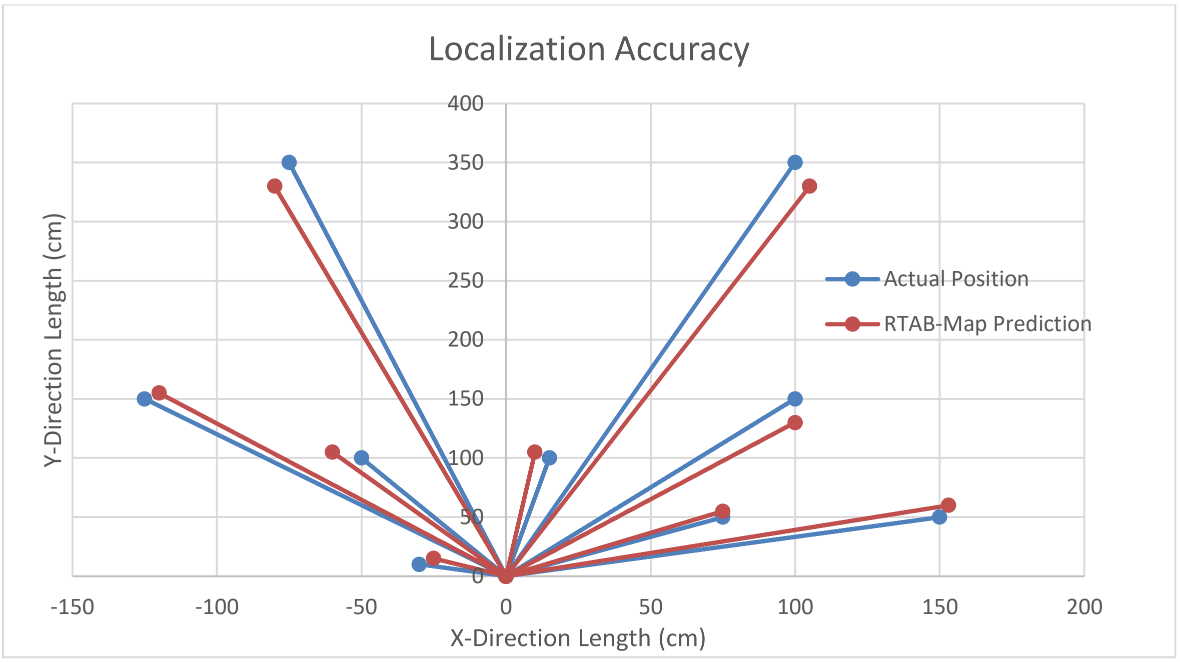


Figure 5.5 Results of the second test

5.3 Autonomous Navigation

When the map of the environment and localization pose of the robot is ready, the move_base node is launched to perform autonomous navigation. The mode is changed to Autonomous navigation where the velocity commands are given by navigation stack. A navigation goal is given with the help of rviz tool. As shown in figure 5-6, navfn creates a global path from initial pose to the goal position. The local planner creates a plan which follows the global plan perceiving any dynamic obstacles in the path.

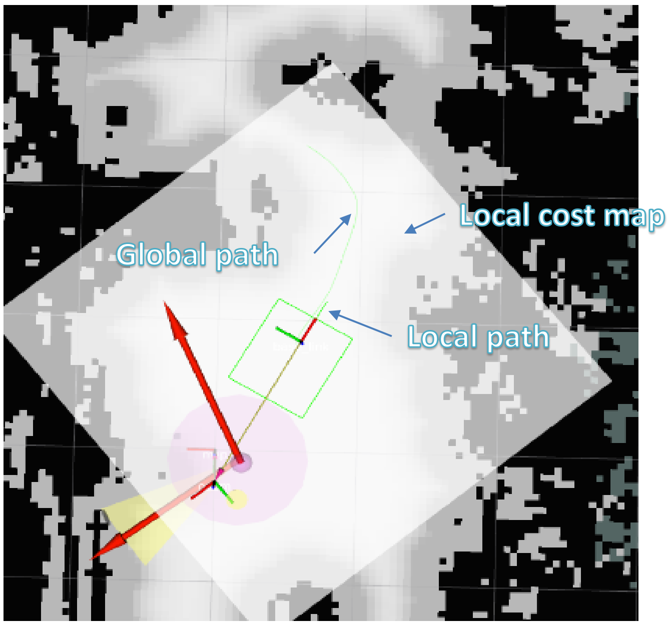


Figure 5.6 Cost maps and planners

A test is made to see if the navigation stack is able to detect dynamic obstacles and change its trajectory accordingly. For this, a map of the environment is created using the RTAB-Map and the robot is localized in the environment. The map and localization pose are loaded to the navigation package. A goal pose in the form of geometric_msgs/poseStamped is given to the robot using RVIZ visualization tool. The navigation stack creates a global and local path for the robot and velocity commands are given accordingly to the robot. While the robot starts moving in the environment, an obstacle is placed in front of the robot.

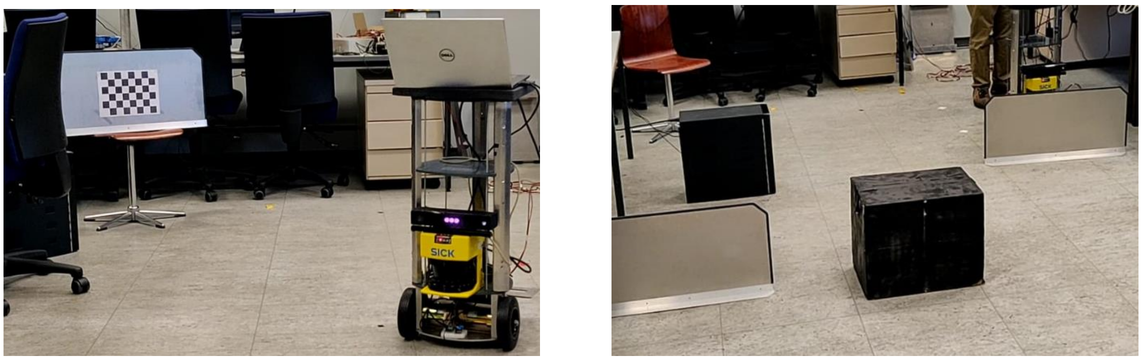


Figure 5.7 Mapping with static obstacles (left), Dynamic obstacles are placed (right)

There are two different configurations to extract Dynamic obstacle information:

- Rtabmap has a nodelet “Rtabmap/obstacle_detection” which is responsible for extracting obstacle information from the sensors by subscribing to point cloud topics.

Then it publishes the obstacle information in the form of occupancy grid through obstacles topic. This was given as input to the move_base node which is responsible for local and global path planning.

- The map which is in ".db" format is loaded to map_server node and it publishes the map in a latched topic /map. This is an occupancy grid map containing the boundary information and static obstacle information in the environment. This is given as input to the move_base node. The localization pose is obtained by rtabmap node (localization mode) and the obstacle information is obtained by subscribing to the sensor cloud topics. This gives the obstacle cost information to the planners in the navigation stack.

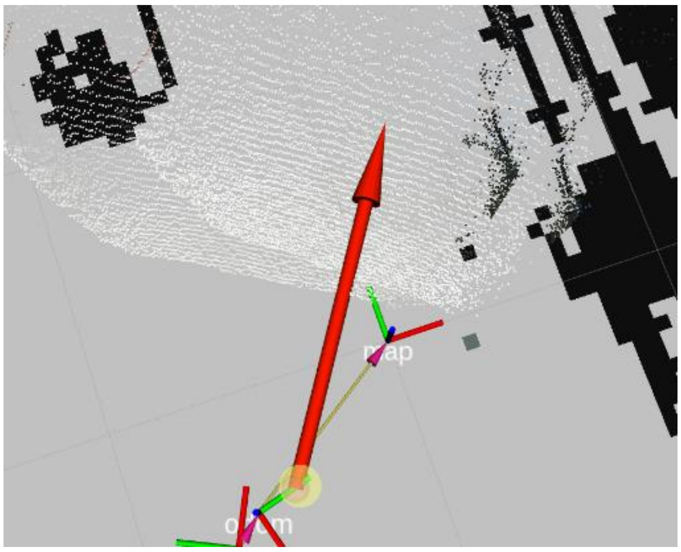


Figure 5.8 Costmap not updated for Dynamic obstacles

During tests, the costmap_2d package of navigation stack was unable to get the obstacle information from the sensor (Kinect) and to update the costmap with the obstacle using above configurations. Which makes the robot to collide with the obstacle. Thus, detection of dynamic obstacle is failing with the used system architecture and implementation.

6 Discussion and Conclusion

In this project, a Visual-SLAM based autonomous navigation system is implemented for TOM3D-Robot using the ROS framework. The hardware configuration and software of the system has been successfully implemented. The accuracy of localization and the performance of the navigation system are tested. The accuracy of the localization is within the range of 90 to 95% as shown in Figure 5.4 and 5.5.

Another important aspect of the project is utilizing a 3D sensor for mapping and localization. A 3D map that can represent the actual environment is built with high accuracy and used for estimating the Odometry. Localization attained from visual odometry showed more accuracy compared to the old system that uses only wheel odometry.

The field of view of Kinect camera is very wide and a lot of memory is consumed to track all the features and map them. Moreover, if there is any change in the environment while mapping, the loop closure is not found and mapping has to be restarted.

Future work

As future work, emphasis has to be given to dynamic obstacles. With regard to mapping, the point cloud data from a 2D laser can be fused with that of Kinect camera to provide a much accurate representation of the environment. With regard to localization, the odometry data obtained from wheel encoders or IMU can be fused with that from visual odometry. This can further increase the accuracy of localization. With these enhancements, this system architecture can become more robust and can autonomously navigate with more accuracy.

7 Bibliography

[1] A. M. B. Pradeep, „Construction of a 3D Map of Indoor Environment,“ *6th International Conference on Smart Computing and Communications, ICSCC 2017,* 2017.

[2] W. D. Rencken, „Autonomous sonar navigation in indoor, unknown and unstructured environments,“ *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94), Munich, Germany, 1994,* Bd. 1, pp. 431-438.

[3] Kortenkamp et al., „Adjustable Autonomy for Human-Centered Autonomous systems,“ 1998.

[4] W. B. T. Dieter Fox, „Probabilistic Methods for Mobile Robot Mapping,“ Proc. of the IJCAI-99 Workshop on Adaptive Spatial Representations of Dynamic Environments, 1999.

[5] „Using EM to Learn 3D Models of Indoor Environments with Mobile Robots,“ *Eighteenth International Conference on Machine Learning,* Bd. 1, pp. 329-336, 2001.

[6] S. D. L. a. J. L. Se, „Vision-based mobile robot localization and mapping using scale-invariant features,“ *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164).,* Bd. IEEE, pp. 2051-2058.

[7] S. Thrun, „ Learning metric-topological maps for indoor mobile robot navigation. Artificial Intelligence,“ *Artificial Intelligence, 99(1),* pp. 21-71, 1998.

[8] K. a. Byun.

[9] K. a. Carpin, „Multi-robot surveillance: An improved algorithm for the GRAPH-CLEAR problem,“ IEEE International Conference on Robotics and Automation, 2008.

[10] D. &. R. R. P. Portugal, „Extracting Topological Information from Grid Maps for Robot Navigation,“ *ICAART (1),* pp. 137-143, 2012.

[11] L.-P. a. Wesley, „An algorithm for planning collision-free paths among polyhedral obstacles,“ 1979.

[12] R. Siegwart, Mobile Robot Localization.

[13] P. Zarchan und H. Musoff, Fundamentals of Kalman Filtering: A Practical Approach, American Institute of Aeronautics and Astronautics, Incorporated, 2000.

[14] H. &. T. S. Ishiguro, „Image-based memory of environment.,“ *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS'96,* Bd. 2, pp. 634-639.

[15] F. B. A. K. M. B. P. F. M. F. T. &. J. L. Duchoň, „Path planning with modified a star algorithm for a mobile robot,“ *Procedia Engineering,* pp. 59-69, 2014.

[16] L. S. P. &. O. M. H. Kavraki, „Probabilistic roadmaps for path planning in high-dimensional configuration spaces,“ 1994.

[17] R. Siddiqui, „Path Planning Using Potential Field Algorithm“.*Medium.*

[18] C. Stachniss, „SLAM,“ [Online]. Available: http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/slam10-fastslam-4.pdf.

[19] S. a. M. M. Thrun, „The graph SLAM algorithm with applications to large-scale mapping of urban structures,“ *The International Journal of Robotics Research 25, no. 5-6,* pp. 403-429, 2006.

[20] J. Collier, „Seventh Canadian Conference on Computer and Robot Vision (CRV 2010),“ Ottawa.

[21] R. M. J. M. M. &. T. J. D. (. Mur-Artal, „ORB-SLAM: a versatile and accurate monocular SLAM system,“ *IEEE transactions on robotics, 31(5),* pp. 1147-1163.

[22] M. L. a. F. Michaud, „RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation,“ *Field Robotics,* Bd. 36, pp. 416-446.

[23] B. a. Kaehler, „Learning OpenCV. Computer vision with OpenCV library,“ 2008.

[24] M. a. F. M. Labbé, „RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation,“ *Journal of Field Robotics 36.2,* pp. 416-446, 2019.

[25] O. S. R. Foundations, „Dwa_local_planner,“ Open Source Robotic Foundations, [Online]. Available: http://wiki.ros.org/dwa_local_planner.

[26] H. Nagaraja, „SLAM-Based Navigation of an Autonomous Mobile Robot Using a 2D LIDAR and ROS,“ University of Siegen, 2018.

[27] H. Nagaraja, „SLAM-Based Navigation of an Autonomous Mobile Robot Using a 2D LIDAR and ROS,“ University of Siegen, 2018.

[28] M. Santala, „3D Content Capturing and Reconstruction Using Microsoft Kinect Depth Camera,“ 2012.

[29] O. S. R. Foundation, „ROS,“ Open Source Robotic Foundation, 2007. [Online]. Available: http://wiki.ros.org/move_base.

[30] R. M. J. J. Mur-Artal, „ORB-SLAM: a versatile and accurate monocular SLAM system,“ *IEEE Transactions on Robotics,* Bd. 31, pp. 1147-11636, 2015.