Project Report

Configuring DJI RoboMaster for ROS Navigation Stack to generate map and navigate autonomously

Course: Mobile Robotics CS-593/ EE 565

Group Members

Syed Muhammad Khalid |20060015

Syed Muhammad Alam | 21060007

Syed Meharullah |21060009

Table of Contents

Table of figures

1. Introduction

A mobile robot is a machine controlled by software that use sensors and other technology to identify its surroundings and move around its environment. Alternatively, mobile robots can rely on guidance devices that allow them to travel a pre-defined navigation route in relatively controlled space. The components of a mobile robot are a controller, sensors, actuators and power system. The controller is generally a microprocessor, embedded microcontroller or a personal computer (PC). An autonomously guided robot knows at least some information about where it is and how to reach various goals and or waypoints along the way. "Localization" or knowledge of its current location, is calculated by one or more means, using sensors such motor encoders, vision, Stereopsis, lasers and global positioning systems. Positioning systems often use triangulation, relative position and/or Monte-Carlo/Markov localization to determine the location and orientation of the platform, from which it can plan a path to its next waypoint or goal. It can gather sensor readings that are time- and location-stamped. Maps are commonly used for robot navigation (e.g., localization). To acquire a map, robots must possess sensors that enable it to perceive the outside world. Sensors commonly brought to bear for this task include cameras, range finders using sonar, laser, and infrared technology, radar, tactile sensors, compasses, and GPS. However, all these sensors are subject to errors, often referred to as measurement noise. More importantly, most robot sensors are subject to strict range limitations. For example, light and sound cannot penetrate walls. These range limitations make it necessary for a robot to navigate through its environment when building a map. The motion commands (controls) issued during environment exploration carry important information for building maps, since they convey information about the locations at which different sensor measurements were taken. Similarly, Path planning algorithms are used by mobile robots, unmanned aerial vehicles, and autonomous cars in order to identify safe, efficient, collision-free, and least-cost travel paths from an origin to a destination. Taken together, a mobile robot with capabilities of localization, mapping and path planning can be used to navigate through obstacles and follow a certain trajectory to reach a goal point.

1.1 Problem Statement

In this project, we attempt to configure Robomaster EP-Core Robot (details in section 2) in ROS to 1) create a map of an environment using mapping algorithm 2) Navigation the robot to avoid obstacles a reach a goal point in the mapped environment.

1.2 Literature Review

Since the 1990s, the field of robot mapping has been dominated by probabilistic techniques. A series of seminal papers by Smith, Self, and Cheeseman [1] introduced a powerful statistical framework for simultaneously solving the mapping problem and the induced problem of localizing the robot relative to its growing map. Since then, robotic mapping has commonly been referred to as SLAM or CML, which is short for simultaneous localization and mapping [2], [3], and concurrent mapping and localization [4], [5], respectively. One family of probabilistic approaches employ Kalman filters to estimate the map and the robot location [6], [7]. The resulting maps usually describe the location of landmarks, or significant features in the

environment, although recent extensions exist that represent environments by large numbers of raw range measurements. An alternative family of algorithms is based on Dempster's expectation maximization algorithm [8], [9]. These approaches specifically address the correspondence problem in mapping, which is the problem of determining whether sensor measurement recorded at different points in time correspond to the same physical entity in the real world. A third family of probabilistic techniques seek to identify objects in the environment, which may correspond to ceilings, walls [10], [11], doors that might be open or closed, of furniture and other objects that move. A* and A* planner [12] are used for shortest path evaluation based on the information regarding the obstacles present in the environment, and the shortest path evaluation for the known static environment is a two-level problem, which comprises a selection of feasible node pairs and shortest path evaluation based on the obtained feasible node pairs. Both of the above mentioned criteria are not available in a dynamic environment, which makes the algorithm inefficient and impractical in dynamic environments. To support path planning in dynamic environments, D* [13] and its variants are discussed as efficient tools for quick re-planning in cluttered environments. As D* and its variants do not guarantee solution quality in large dynamic environments, we also explore Rapidly-exploring Random Trees (RRTs) [14] and a hybrid approach combining Relaxed A* (RA*) [15] and one meta-heuristic algorithm. The hybrid approach comprises two phases: the initialization of the algorithm using RA* and a post-optimization phase using one heuristic method that improves the quality of solution found in the previous phase.

2 Project Description:

In our project, we have used the RoboMaster DJ'I intelligent control robot to perform various operations in order to get our desired objectives such as mapping and navigation.

2.1 RoboMaster Ep-Core

After the inception of RoboMaster S1, RoboMaster Ep-core has been introduced in the educational area of robotics that takes the level of learning to a different level. Any educational institute can take benefit of Artificial intelligence and programming from this robot. RoboMaster Ep core consist of various components such as Gripper which helps to move an object from one place to another, Robotic Arm that allows to perform operations despite being away from the target, Sensor Adapter and Power Connector, and IR sensors.

Figure 1: Robomaster EP core [16]

2.2 Robotic Mapping

Robotic mapping has been considered a prominent area of research in robotics and Artificial Intelligence for the last couple of decades. It addresses the problem of three-dimensional models of physical environments via specific model robots such as DJ's. Problem of mapping is observed as one of the essential problems while constructing the perfectly autonomous Mobile Robots. Maps are widely used for the navigation of robots. To perceive a map, a robot must contain number of sensors that gives information about the environment. Various mapping algorithms have been developed over the years which are probabilistic. In our work we have considered the gmapping (SLAM) algorithm.

Figure 2 Map and a robot path

2.3 Gmapping

Gmapping is a particle filter that uses the range sensor data to learn a grid map. Particle filters are used to address the problem of simultaneous localization and mapping problem (SLAM). It takes the laser range data and tf (transformation) as an input data and generates a map.

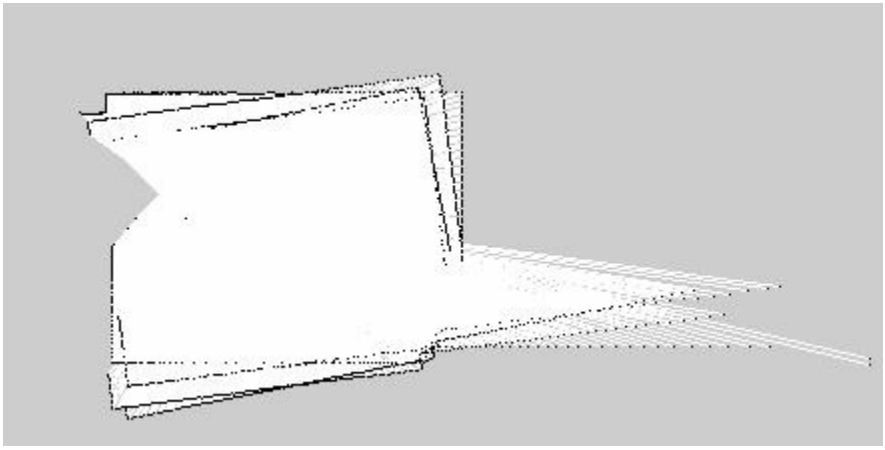

Figure 3: Random gmapping map [17]

2.4 Tf2 Transformation

A robotic mechanism typically carries many coordinate frames that varies over the time, it consists of number of frames called as inertial frame, base frame, joint frame and link frame etc. A tf2 transformation holds the track of the aforementioned frames during an operation and allows the analyst to observe various points. Frames positions with respect to the preceding frames or world frame can be seen or observed via tf2 transformation. It operates in a distributed manner which means that the information about all frames of a robot is available to Robot Operating System (ROS) Components.

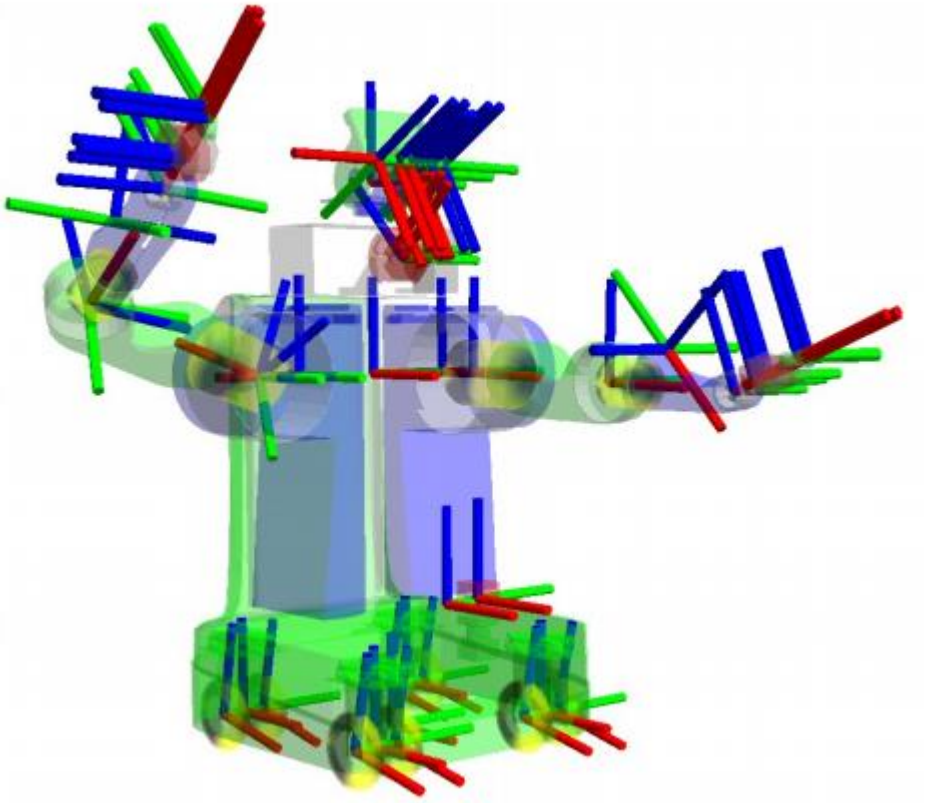

Figure 4: Frames attached to a robot [17]

2.5 Navigation Stack

A navigation stack is a set of algorithms that takes in information from sensors and Odometry and gives an output in the form of velocity. Velocities are further used to operate a mobile robot in an environment. Being a Prerequisite for a navstack use, a robot must be operating on ROS, it must have a tf tree, publish the data taken from sensor and Odometry data using the authentic message type. Navstack requires basic hardware requirements such as it can be used for both holonomic robots and differential drive robots, it assumes that mobile robot is controlled via command velocities. It requires a laser mounted on top of the robot, initially navstack was made for robots that are circular and square for the simplicity.

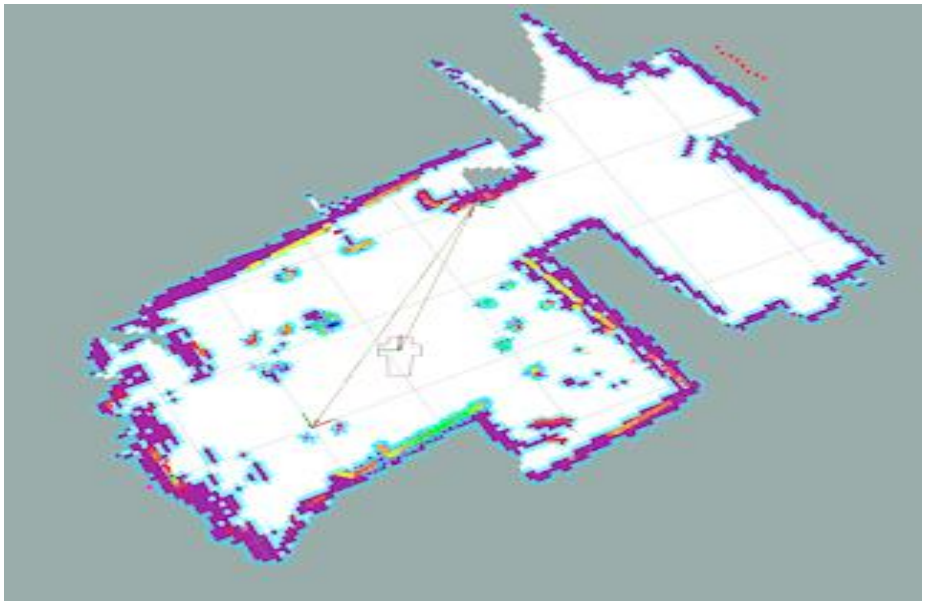

Figure 5 Navigation stack

3 Methodology

The project methodology is divided into three main steps:

1) Configuring the robomaster EP core and the attached IR sensors to replicate a LIDAR sensor. Furthermore, keyboard based input for robot movement was also implemented
2) Configuring the Robomaster to implement Gmapping algorithm. This includes creating scan topics, odometry topics, relevant transformations, and gmapping parameter settings.
3) Implementing ROS navigation stack to autonomously move the robot from initial position to goal position while avoiding obstacles.

3.1 Configuring Robomaster IR sensors

Three IR sensor were placed at 90 degrees difference from each other on the Robomaster robot (Figure 1). The aim was to replicate the Lidar sensor reading based on the three sensors. A sensor type message for ROS LaserScan was used for this purpose.

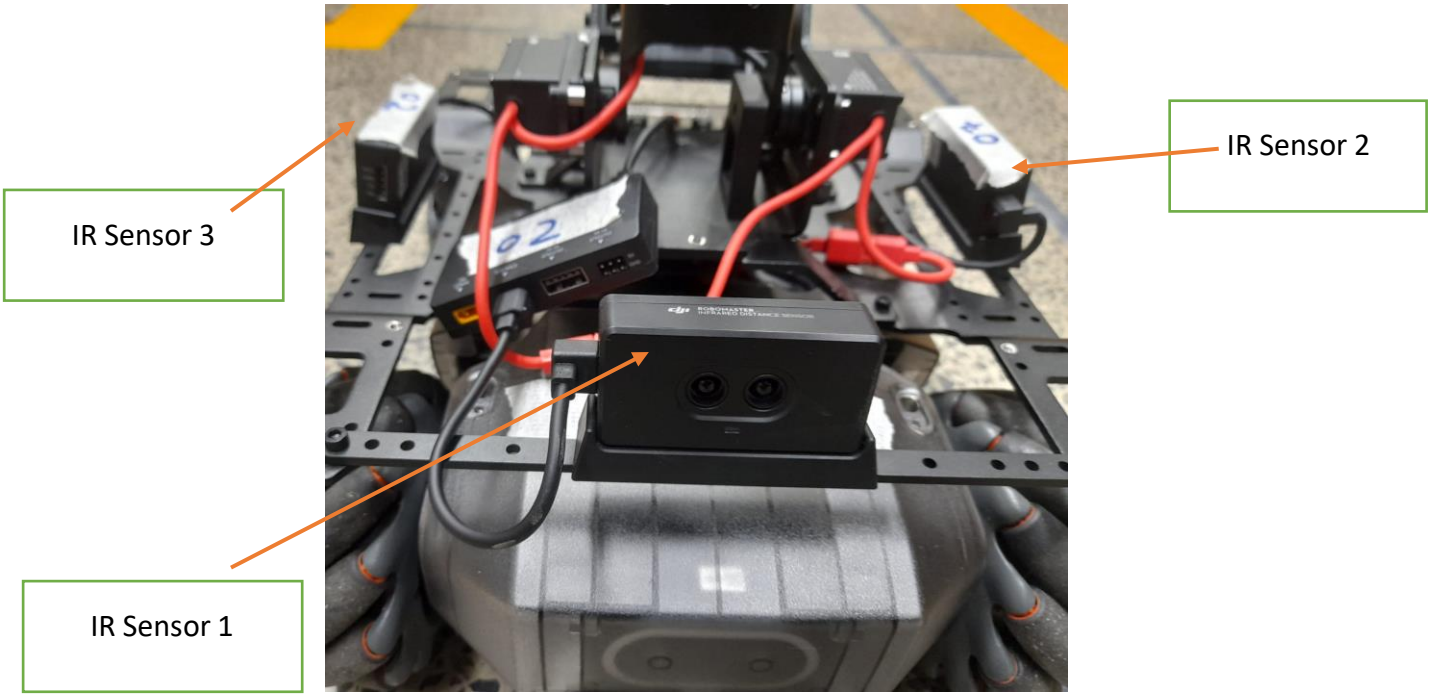

Figure 6: IR sensor placement on Robomaster

The following were the parameters that were included the LaserScan message:

```
scan.header.stamp = current_time
scan.header.frame_id = 'base_link'
scan.angle_min = -1.57
scan.angle_max = 1.57
scan.angle_increment = math.pi / 2
scan.time_increment = (1.0 / laser_frequency) / (num_readings)
scan.range_min = 0.0
scan.range_max = 5.0
scan.scan_time  = (1 / laser_frequency)
scan.ranges = IR sensor1 readings, IR sensor2 readings, IR sensor3 readings
```

Figure 7: Pseudo code for Laser Scan message

3.2 Implementing keyboard movement for Robomaster

In order to move the robot based on keyboard commands, pynput keyboard python library was used. This allowed the program to wait for a keyboard input from the user and implement the action based on the key pressed. In order to increase the accuracy for our movement, we move the robot 0.05 m at one linear movement command, and angle of 5 degree at one angular movement command.

```
if event.key == keyboard.KeyCode.from_char('w'):
    ep_chassis.move(x=0.05, y=0, z=0, xy_speed=0.5).wait_for_completed()
elif event.key == keyboard.KeyCode.from_char('s'):
 ep_chassis.move(x=-0.05, y=0, z=0, xy_speed=0.5).wait_for_completed()
elif event.key == keyboard.KeyCode.from_char('d'):
    ep_chassis.move(x=0, y=0, z=-5, xy_speed=0.5).wait_for_completed()
elif event.key == keyboard.KeyCode.from_char('a'):
    ep_chassis.move(x=0, y=0, z=5, xy_speed=0.5).wait_for_completed()
elif event.key == keyboard.KeyCode.from_char('q'):
    ep_chassis.move(x=0.4, y=0, z=5, xy_speed=0.5).wait_for_completed()
elif event.key == keyboard.KeyCode.from_char('z'):
    ep_chassis.move(x=-0.4, y=0, z=5, xy_speed=0.5).wait_for_completed()
elif event.key == keyboard.KeyCode.from_char('l'):
 break
```

Figure 8: Pseudo code keyboard movement

3.3 Gmapping Implementation

Gmapping (SLAM) was used to create a map of an environment. The gmapping algorithm requires scan data and transformation data as input as outputs a mapped output based on these. ROS provides a system called tf2 (Transform version 2) to handle these transformations for us. Any node can use the tf2 libraries to broadcast a transform from one frame to another. As mentioned above, these transforms will need to form a tree structure, where each frame is defined by one (and only one) transform from another frame, but can have any number of frames dependent on it. In order to provide transformation 2 different type of transformers were used, tf2 transformer and tf2 static transformers. This was used to cater for static transformation and dynamic transformation.

The transformation for each frame requires parameters for translational and rotation. The following is the pseudo code for transformation tf2.

```
static_transformStamped_lb.header.stamp = rospy.Time.now()
static_transformStamped_lb.header.frame_id = "base_link"
static_transformStamped_lb.child_frame_id = 'base_laser'
static_transformStamped_lb.transform.translation.x = Linear x translation
static_transformStamped_lb.transform.translation.y = Linear x translation
static_transformStamped_lb.transform.translation.z = 0.0
static_transformStamped_lb.transform.rotation.x = Quaternion x
static_transformStamped_lb.transform.rotation.y = Quaternion  y
static_transformStamped_lb.transform.rotation.z = Quaternion  z
static_transformStamped_lb.transform.rotation.w = Quaternion  w
```

Figure 9: Pseudo code for tf2 transformation

Individual transformation is described in the figure 5. There are a total 6 frames with odom to base-link frame constantly changing with change in robot's motion. Other 5 frames relation are static for simplicity.

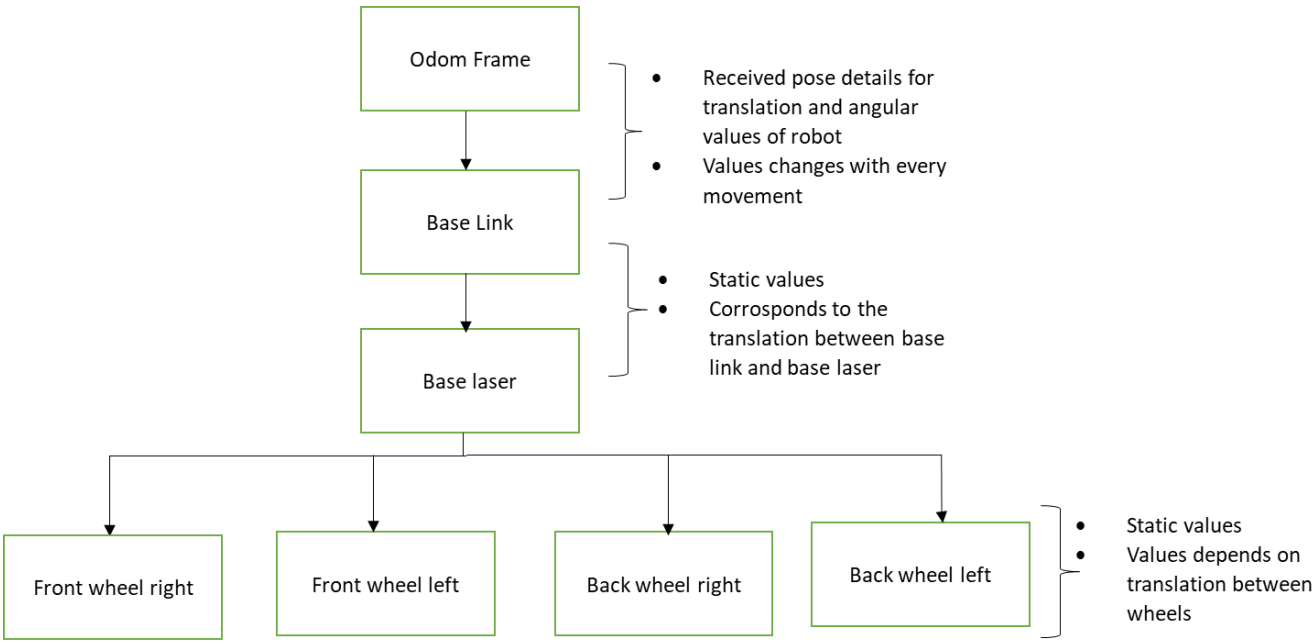

Figure 10: Transformation frames in tf2

Similarly, several of gammping parameters were set in order to achieve high quality results. Following are the details of some if the important parameters.

- maxRange (float): The maximum range of the sensor.
- Occ_thresh (float, default: 0.25): Threshold on gmapping's occupancy values
- Particles (int, default: 30): Number of particles in the filter
- LinearUpdate (float, default: 1.0): Process a scan each time the robot translates this far
- AngularUpdate (float, default: 0.5): Process a scan each time the robot rotates this far
- MinimumScore (float, default: 0.0): Minimum score for considering the outcome of the scan matching good. Can avoid jumping pose estimates in large open spaces when using laser scanners with limited range
- Ogain: Gain to be used while evaluating the likelihood, for smoothing the resampling effects
- srr : Odometry error in translation as a function of translation (rho/rho)
- ~srt (`float`, default: 0.2): Odometry error in translation as a function of rotation (rho/theta)
- ~str (`float`, default: 0.1): Odometry error in rotation as a function of translation (theta/rho)

We set the following parameters for our gmapping:

- base_frame value="base_link"
- odom_frame value="odom"
- map_update_interval" value="5.0"/>
- maxUrange value="4.0"/>
- maxRange value="5.0"/>
- sigma value="0.05"/>
- kernelSize" value="1"/>
- lstep" value="0.05"/>
- astep" value="0.05"/>
- iterations" value="5"/>
- lsigma" value="0.075"/>
- ogain" value="3.0"/>
- lskip" value="0"/>
- minimumScore" value="100"/>
- srr" value="0.01"/>
- srt" value="0.02"/>
- str" value="0.01"/>
- stt" value="0.02"/>
- linearUpdate" value="0.00"/>
- angularUpdate" value="0.00"/>
- temporalUpdate" value="-1.0"/>
- resampleThreshold" value="0.5"/>
- particles" value="80"/>
- xmin" value="-50.0"/>
- ymin" value="-50.0"/>
- xmax" value="50.0"/>
- ymax" value="50.0"/>
- xmin" value="-1.0"/>
- ymin" value="-1.0"/>
- xmax" value="1.0"/>
- ymax" value="1.0"/>
- delta" value="0.05"/>
- occ_thresh" value="0.75"/>
- llsamplerange" value="0.01"/>
- llsamplestep" value="0.01"/>
- lasamplerange" value="0.005"/>
- lasamplestep" value="0.005"/>

3.4 Navstack Implementation

Navigation stack that takes in information from odometry, last data, and a goal pose and outputs safe velocity commands that are sent to a mobile base. First objective was to create odometry message that is published from our main node in ROS.

```
odom.child_frame_id = "base_link"

odom.pose.pose = Pose(Point(x, y, 0.), Quaternion(*odom_quat))

odom.twist.twist = Twist(Vector3(0.5, 0.5, 0), Vector3(0, 0, 5))
```

Figure 11: Odom message type

Once our odom data is set, we created a velocity command subscriber that is published nav stack. These commands are provided in the callback function which are set as the velocity of angular speed for the robot. The robot subscribes to the "cmd_vel" (i.e. velocity command) topic that takes velocities and converts them into motor commands.

Following are the details of navstack algorithm set:

The ROS Navigation Stack uses two costmaps to store information about obstacles in the world.

1. **Global costmap:** This costmap is used to generate long term plans over the entire environment….for example, to calculate the shortest path from point A to point B on a map.
2. **Local costmap:** This costmap is used to generate short term plans over the environment….for example, to avoid obstacles.

Following are the parameters for the costmap:

- global_frame: odom
- update_frequency: 30.0
- publish_frequency: 30.0
- transform_tolerance: 0.2
- resolution: 0.1
- Obstacle range: 4
- Ray trace range: 5

In addition to the costmap configurations we did in the previous section, we need to configure ROS Navigation Stack's base local planner. The base_local_planner computes velocity commands that are sent to the robot base controller. The base_local_planner package provides a controller that drives a mobile base in the plane. This controller serves to connect the path planner to the robot. Using a map, the planner creates a kinematic trajectory for the robot to get from a start to a goal location. Along the way, the planner creates, at least locally around the robot, a value function, represented as a grid map. This value function encodes the costs of traversing through the grid cells. The controller's job is to use this value function to determine dx,dy,dtheta velocities to send to the robot.

The basic idea of the Dynamic Window Approach (DWA) algorithms is as follows:

1. Discretely sample in the robot's control space (dx,dy,dtheta)
2. For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
3. Evaluate (score) each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard illegal trajectories (those that collide with obstacles).
4. Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
5. Rinse and repeat.

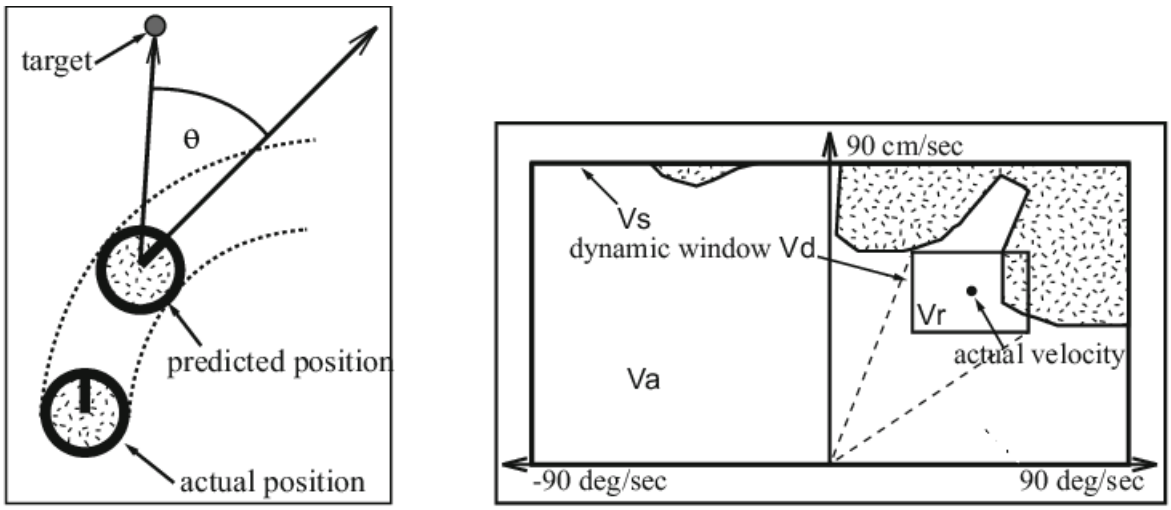

Figure 12: DWA algorithm illustration [18]

3.5 Path planning

In order to score trajectories efficiently, navstack uses the concept of Map Grid is used. For each control cycle, a grid is created around the robot (the size of the local costmap), and the global path is mapped onto this area. This means certain of the grid cells will be marked with distance 0 to a path point, and distance 0 to the goal. A propagation algorithm then efficiently marks all other cells with their manhattan distance to the closest of the points marked with zero. The principle of DWA local planning is the search for a suitable local plan in every control cycle. For that purpose, a number of candidate trajectories is generated. For a generated trajectory, it is checked whether it collides with an obstacle. If not, a rating is given to compare several trajectories picking the best.

3.6 Autonomous navigation

The configuration files will be used by ROS Navigation Stack's move_base node. The move_base node is the work horse behind the scenes that is responsible for planning a collision-free path from a starting location to a goal location for a mobile robot.

The move-base node subscribes to the following topics:

- **/move_base_simple/goal :** Goal position and orientation (geometry_msgs::PoseStamped). The messages on this topic are generated using the goal button on RViz.

The publisher will **publish** to the following topics:

- **/cmd_vel** : Linear and angular velocity command (geometry_msgs/Twist Message)

AMCL Configuration

The ROS Navigation Stack requires the use of AMCL (Adaptive Monte Carlo Localization), a probabilistic localization system for a robot. AMCL is used to track the pose of a robot against a known map. It takes as input a map, LIDAR scans, and transform messages and outputs an estimated pose

The amcl node subscribes to the following topics:

- **/scan :** Laser scan messages from the LIDAR (sensor_msgs/LaserScan).
- **/tf** : Coordinate frame transformations (tf/tfMessage).
- **/initialpose** : The initial position and orientation of the robot using quaternions. (geometry_msgs/PoseWithCovarianceStamped) — RViz initial pose button publishes to this topic.
- **/map :** The occupancy grid map created using gmapping, Hector SLAM, or manually using an image (nav_msgs/OccupancyGrid).

The amcl node will **publish** to the following topics:

- **/amcl_pose :** Robot's estimated pose in the map (geometry_msgs/PoseWithCovarianceStamped).
- **/particlecloud:** The set of pose estimates being maintained by the filter (geometry_msgs/PoseArray).
- **/tf** (tf/tfMessage): Publishes the transform from odom (which can be remapped via the ~odom_frame_id parameter) to map.

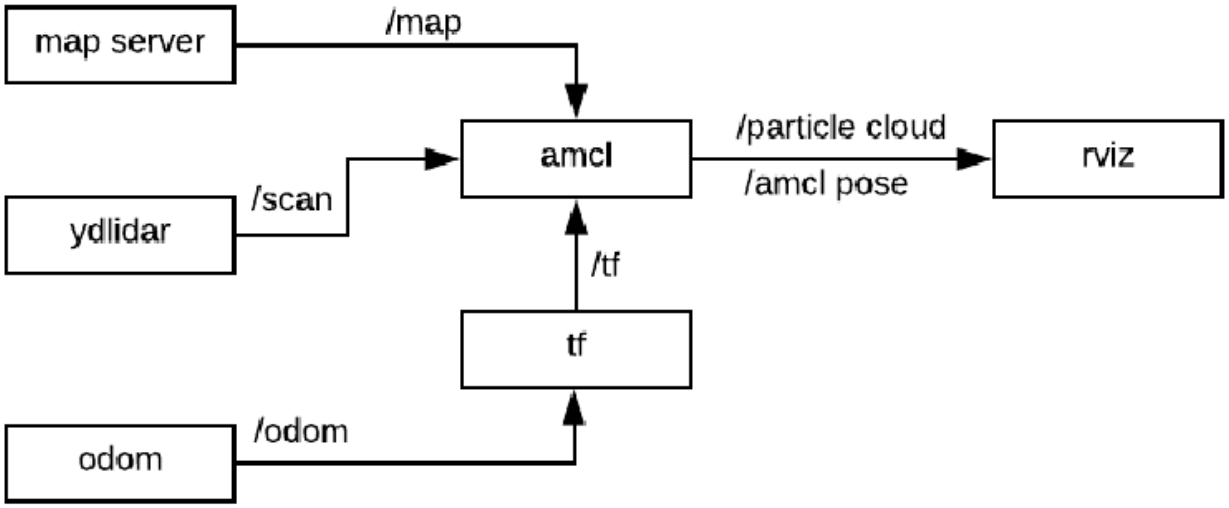

Figure 13: AMCL topics publishers and subscribers

4 Results

4.1 Laser Scan data

The first objective of the project was replicate the 3 IR sensors to perform like a Lidar Sensor. The configuration used in the methodology allowed 3 Laser data to publish:

1) Before a move command was given
2) During the movement
3) After the movement is completed

Once published the IR sensor data is appended to Scan ranges array, which then came be viewed through RVIZ. The figure 14 shows the result of a single Laser Scan topic published. Figure 15 shows the result when the robot is moved in a circular motion

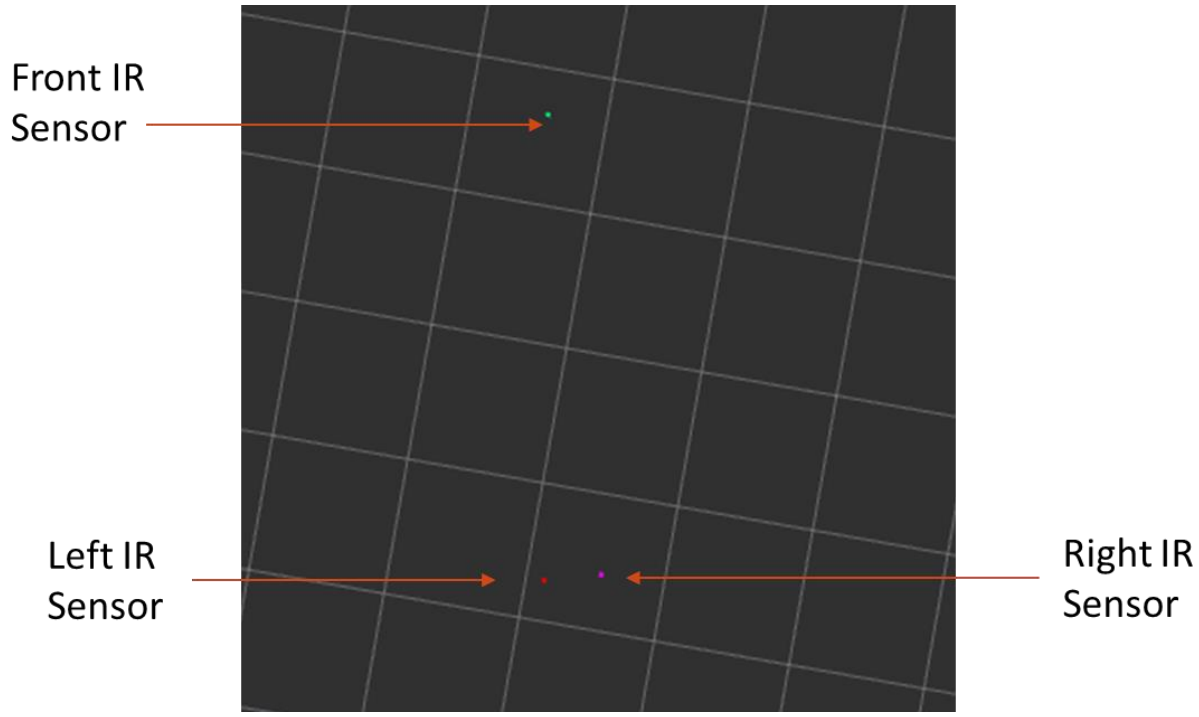

Figure 14: Result of Laser Scan data on Rviz

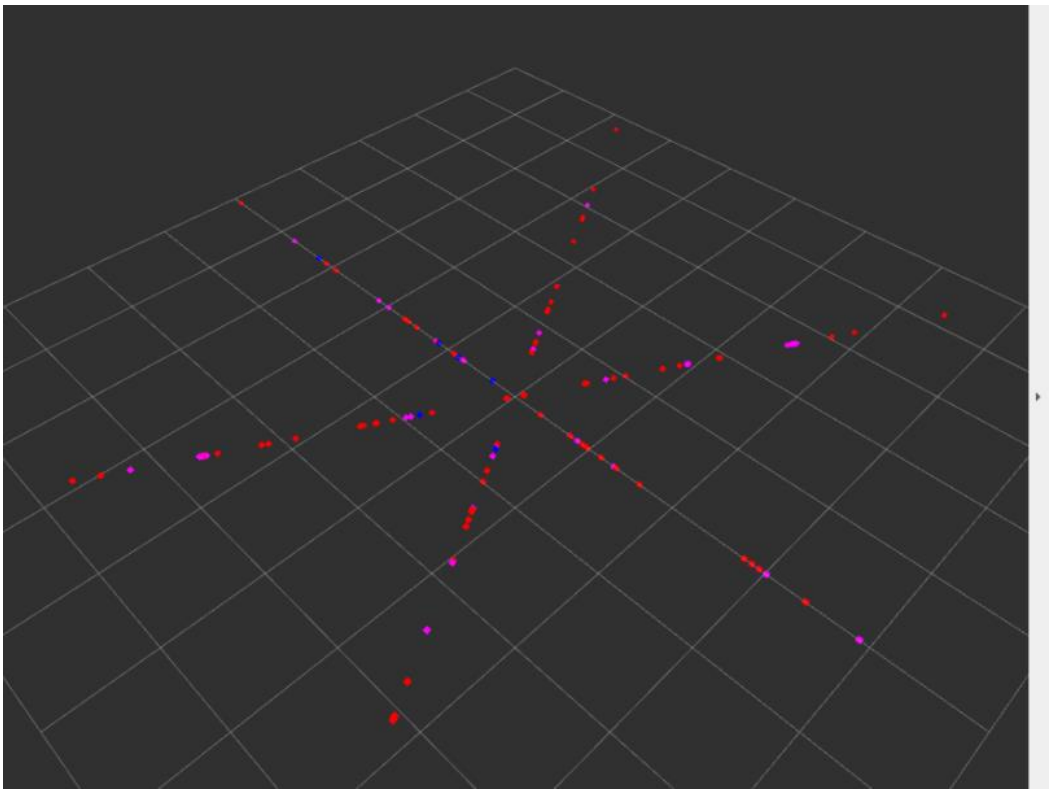

Figure 15: LaserScan resut on Rviz when moved in circular option

4.2 Transformation

The 2nd objective was to add transformation between different frames of the robot. This was done considering the base link and base laser relation with odom and then the 4 wheels of the robot. The figure below shows the rqt tree that explains the relation between transformation frames.

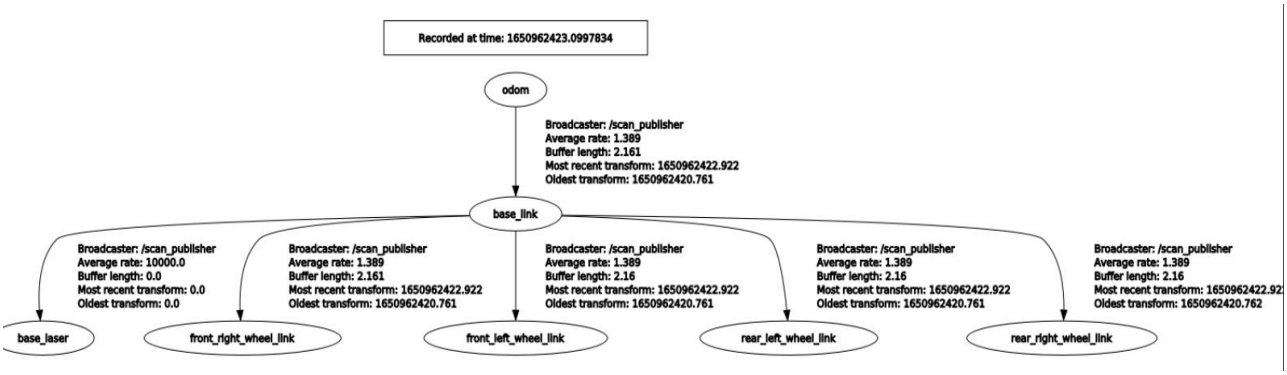

Figure 16: Rqt frames showing transformation frames

4.3 Gmapping implementation

With the transformation and scan data information being published, the gmapping takes in the information and publishes a map topic. Figure 17 shows the transformation frames and topics being published after gmapping node is enabled. Figure 18 shows the map created when robot is moving 5-10 degree with zero linear velocity.

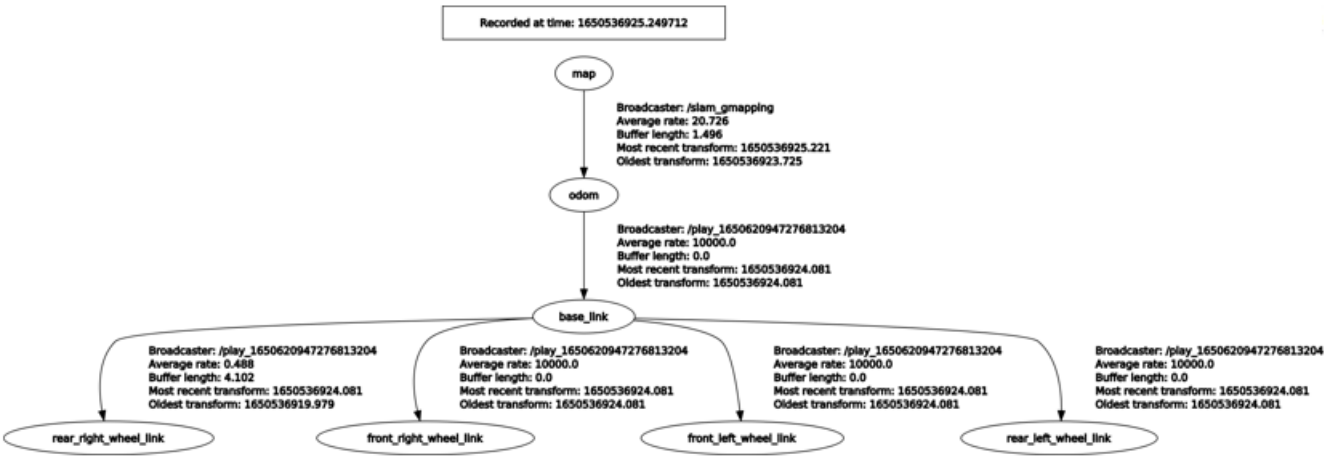

Figure 17: Frames transformation after gmapping node is enabled

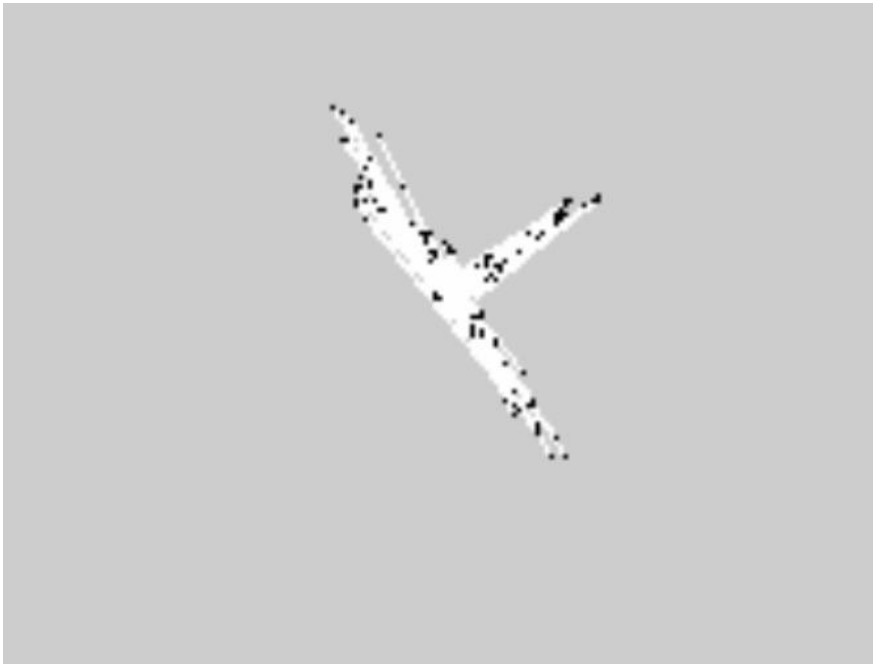

Figure 18: Gmapping map created with small angle changes

4.4 Gmapping testing

In order to make the map created more accurate, several parameters of gmapping were tested and changed. Occ threshold, max range, Ogain, linear update and angular update (See section 3 for more detail) were changed. The figure 19 shows the result of a map created with a wall at the right hand side of the robot. Figure 20 shows the result of the gmapping map when the robot is rotated only.

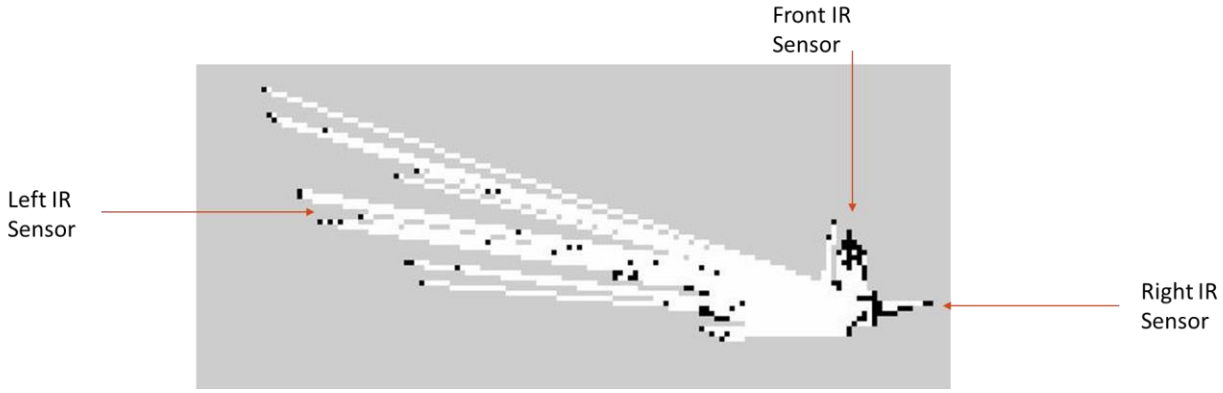

Figure 19: Map created with a wall at right sensor

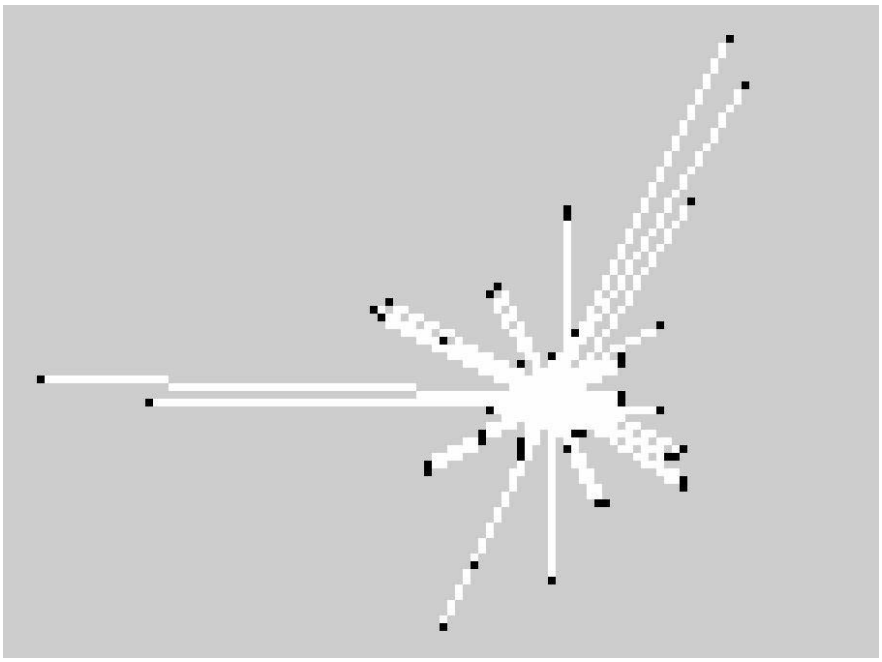

Figure 20: Map created when robot is rotated

4.5 Map of arena

The final of the arena was created with robot moving at very slow sped with no turns. It was found out that the robot missed some readings on the map due to its limitation of 3 IR sensor.

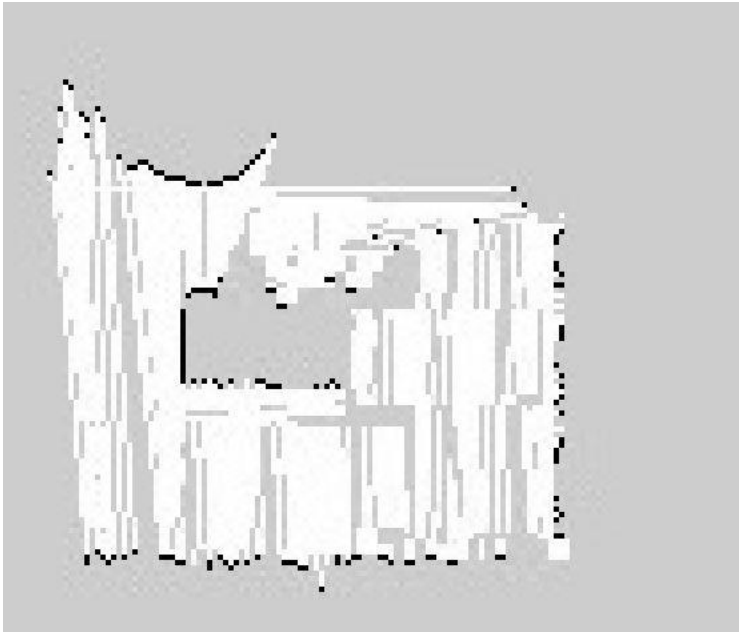

Figure 21: Final map created of the Arena

4.6 Navstack implementation

Move base launch file was enabled and robot connection was made. The navstack was able to find pose array and map topics. Figure below shows the result in rviz when move based launch was enabled.

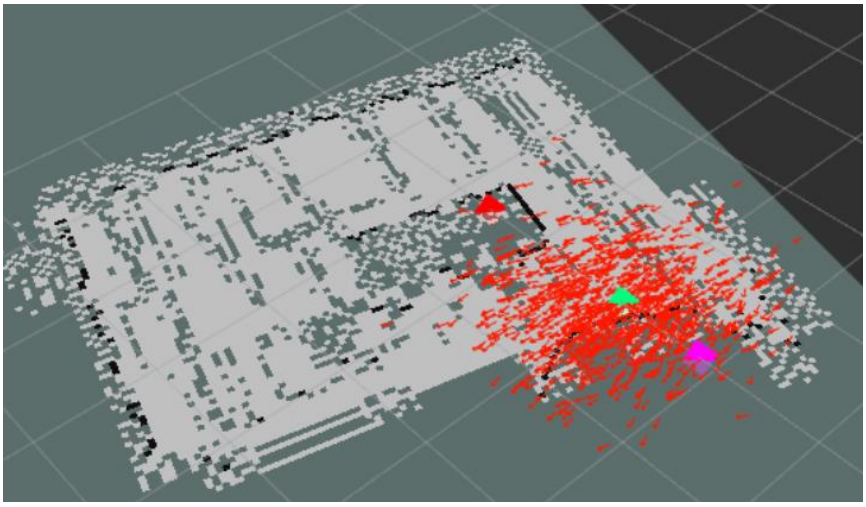

Figure 22: RVIZ result with map, pose array and laser scan when move based node is enabled

Similarly, the navstack was also able to find local cost map, that corresponds to the obstacles in the real map.

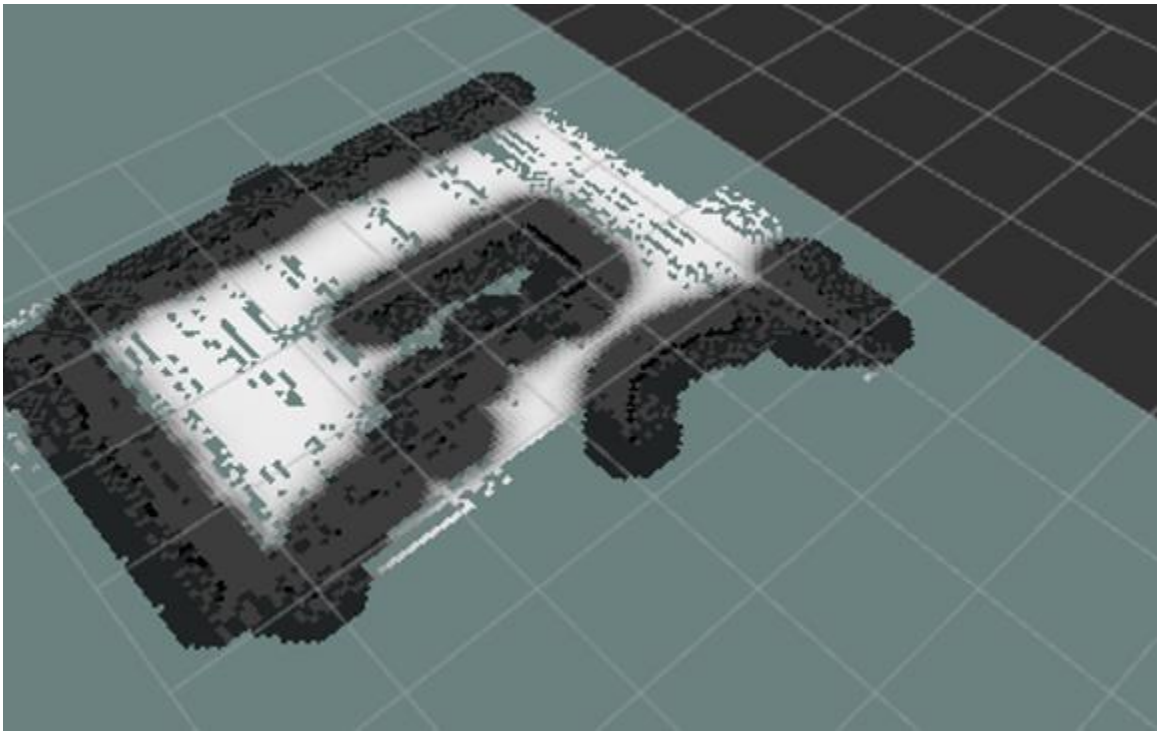

Figure 23: local cost map

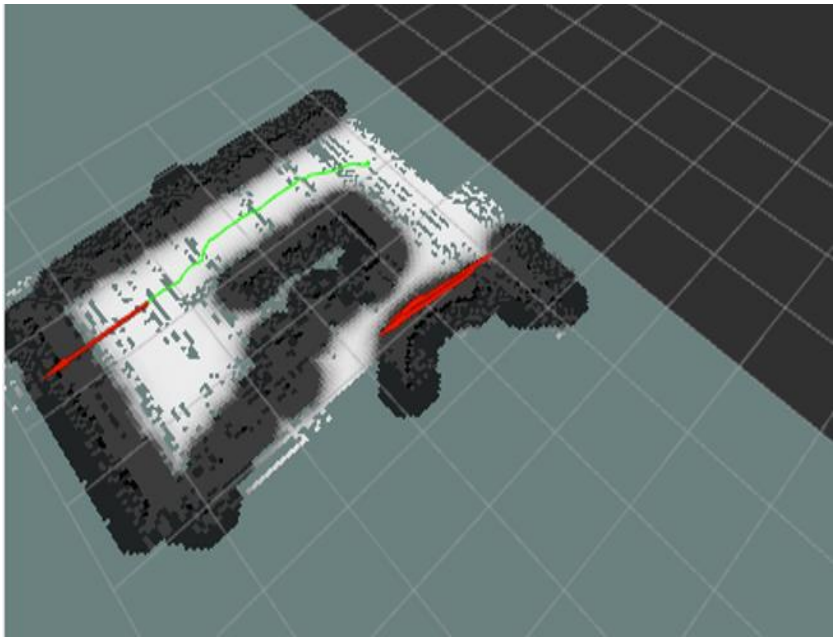

Figure 24: Path planning with local costmap and odometry

4.7 Autonomous navigation

The last objective was autonomously movement of the robot in the arena. In order to perform this, we created a subscriber for command velocity that takes the velocity commands published by the move base node. The figure below shows the local path planner, local cost map and odometry information in rviz along with cmd velocity commands published by move base node.

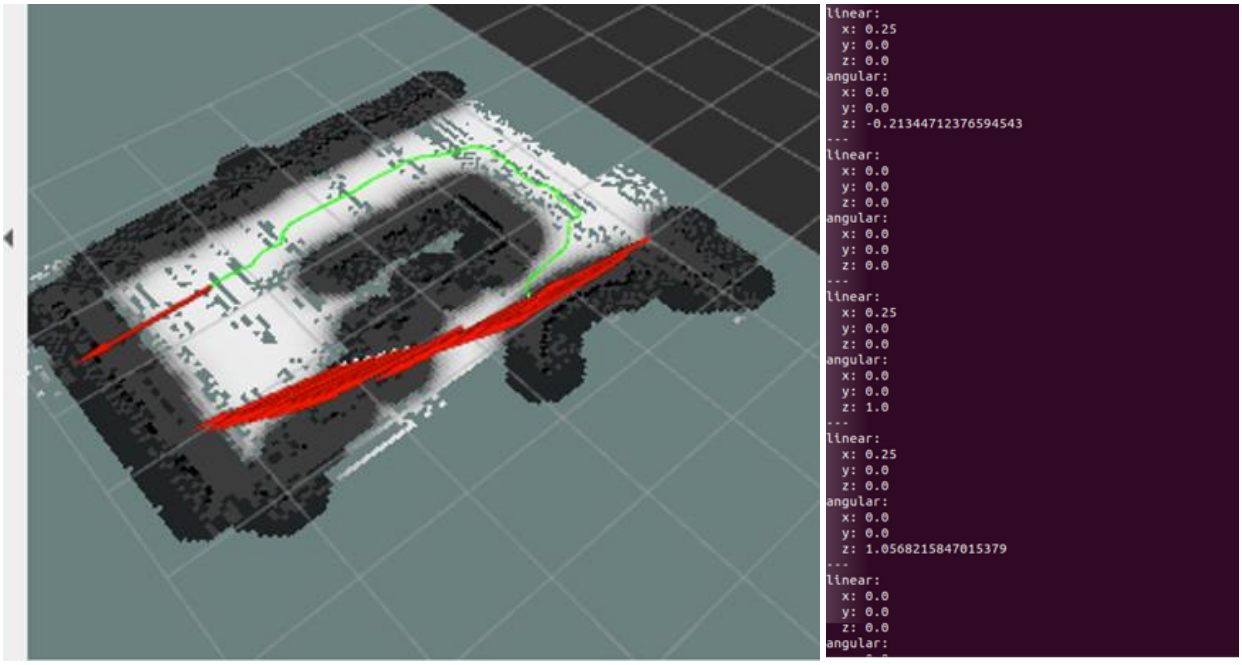

Figure 25: local path planner, local cost map and odometry information in rviz with command velocity data

Similarly, when facing obstacles, the move based published information that an obstacle is detected. Below is screenshot of the message command.

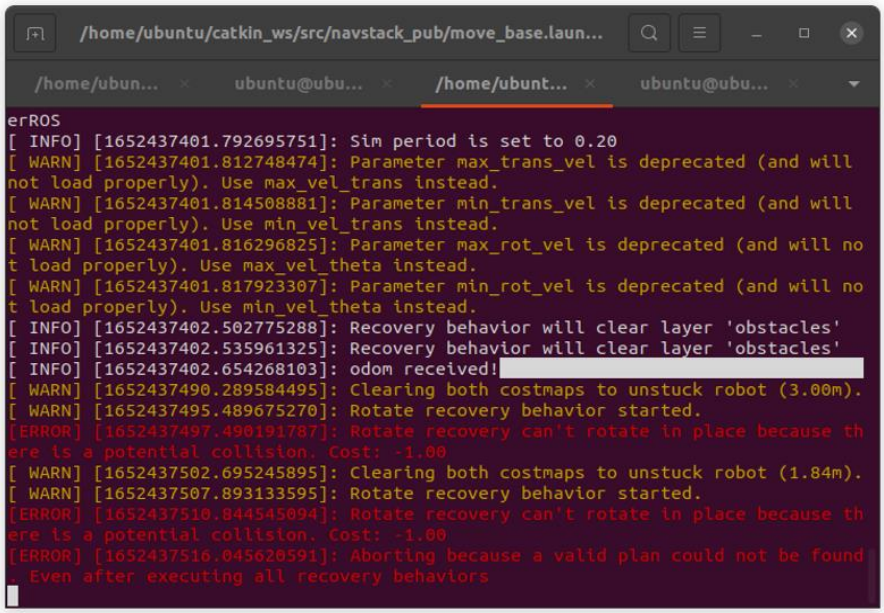

Figure 26: Navstack detected a collision with obstacle

5 Conclusion

In this project, we attempted to autonomously drive robotmaster ep core, on a mapped environment to avoid obstacles and replicated a lidar sensor using 3 IR sensor. We showed that it is possible to create a map on gmapping package using only 3 IR sensor. The IR sensor was successfully able to replicate lidar sensor and published the relevant data. The gmapping map was quite accurate considering the limitation of our range and sensor values. The navstack algorithms was successfully able to use information of odometry and map data to create a path that avoided obstacles. The results published by navstack was not successfully implemented on the robot, one reason could be the absence of URDF file that would contain the information of our robot model. Few resources that have successfully solved the problem, have used URDF file to move the robot more accurately.

6 References

[1] R. Smith', M. Self^, and P. Cheesemans, "Estimating Uncertain Spatial Relationships in Robotics*."

[2] G. Dissanayake, H. Durrant-Whyte, and T. Bailey, "Computationally efficient solution to the simultaneous localisation and map building (SLAM) problem," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2, pp. 1009–1014, 2000, doi: 10.1109/ROBOT.2000.844732.

[3] H. Durrant-Whyte, S. Majumder, S. Thrun, M. de Battista, and S. Scheding, "A Bayesian Algorithm for Simultaneous Localisation and Map Building," pp. 49–60, 2003, doi: 10.1007/3-540-36460-9_4.

[4] J. J. Leonard, H. Jacob, and S. Feder, "A Computationally Efficient Method for Large-Scale Concurrent Mapping and Localization."

[5] H. Hexmoor and M. M. Matari´c, "A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots," Accessed: May 13, 2022. [Online]. Available: http://www.cs.cmu.edu/thrunhttp://www.cs.uni-bonn.de/wolframhttp://www.cs.uni-bonn.de/fox.

[6] M. Csorba, "Simultaneous Localisation and Map Building."

[7] J. A. Castellanos and J. D. Tardós, "Mobile Robot Localization and Map Building," *Mob. Robot Localization Map Build.*, 1999, doi: 10.1007/978-1-4615-4405-0.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data Via the EM Algorithm," *J. R. Stat. Soc. Ser. B*, vol. 39, no. 1, pp. 1–22, Sep. 1977, doi: 10.1111/J.2517-6161.1977.TB01600.X.

[9] G. J. McLachlan and T. (Thriyambakam) Krishnan, "The EM algorithm and extensions," p. 359.

[10] L. Iocchi, K. Konolige, and M. Bajracharya, "Visually Realistic Mapping of a Planar Environment with Stereo," 2000.

[11] Y. Liu, R. Emery, D. Chakrabarti, W. Burgard, and S. Thrun, "Using EM to Learn 3D Models of Indoor Environments with Mobile Robots."

[12] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968, doi: 10.1109/TSSC.1968.300136.

[13] A. Stentz, "Optimal and Efficient Path Planning for Partially-Known Environments," 1994.

[14] S. M. Lavalle and S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998, Accessed: May 13, 2022. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.1853.

[15] A. Ammar, H. Bennaceur, I. Châari, A. Koubâa, and M. Alajlan, "Relaxed Dijkstra and A* with linear complexity for robot path planning problems in large-scale grid environments," *Soft Comput.*, vol. 20, no. 10, pp. 4149–4171, Oct. 2016, doi:

10.1007/S00500-015-1750-1.

[16] “RoboMaster EP Core-Learning Unleashed-DJI.” https://www.dji.com/robomaster-ep-core (accessed May 14, 2022).

[17] “Documentation - ROS Wiki.” http://wiki.ros.org/Documentation (accessed May 14, 2022).

[18] C. Luo, F. Shen, H. Mo, and Z. Chu, “An improved ant-driven approach to navigation and map building,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10385 LNCS, pp. 301–309, 2017, doi: 10.1007/978-3-319-61824-1_33.