AUTONOMOUS MOBILE ROBOT NAVIGATION WITH GPU ACCELERATION FOR UNMANNED UV-C BASED DECONTAMINATION APPLICATIONS

By

BIRJU JITENDRA VACHHANI

A thesis submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Master of Science

Graduate Program in Mechanical and Aerospace Engineering

Written under the direction of

Qingze Zou

And approved by

_____

_____

_____

New Brunswick , New Jersey

[JANUARY 2019]

ABSTRACT OF THE THESIS

AUTONOMOUS MOBILE ROBOT NAVIGATION WITH GPU ACCELERATION FOR UNMANNED UV-C BASED DECONTAMINATION APPLICATIONS

By BIRJU JITENDRA VACHHANI

Thesis Director :

Qingze Zou

Navigation is a complex robotic problem solving which makes the mobile robot intelligent for decision making in dynamic environments. The objective of this thesis is to achieve autonomous mapping and navigation of a 2D environment for unmanned decontamination and sterilization of rooms in facilities such as hospitals and hotels, where UV-based degermination system is carried by a four-wheeled robot during the navigation. The challenge arises from the intensive online computation needed in the navigation process, as a Light Detection and Ranging (LiDAR) system is employed for Simultaneous Localization and Mapping (SLAM) and obstacle detection, and occupancy grid is employed as the data structure to represent surrounding environments in robotics. However, path planning on an occupancy grid is computationally intensive. For example, path planning on a 25m x 25m grid can involve processing of 250,000 grid cells on a 0.05m resolution grid. In this project, we developed a parallel computation framework to substantially reduce the processing times, and thereby, achieve dynamic obstacle avoidance and autonomous exploration. Specifically, a obstacle inflation module was created using parallel computing. Building on that, a graph based path planner was also developed for autonomous as well as user instructed navigation. The system was thoroughly tested in various static and dynamic indoor environments.

ACKNOWLEDGEMENT

I would like to take this opportunity to thank all the people who made my stay at Rutgers University as a graduate student very pleasurable.

Firstly , I would like to thank my Graduate Adviser Dr. Qingze Zou, for giving me all the necessary support for completing this project smoothly and making the experience that of a great learning. I would also like to thank Dr. Xiaoli Bai and Dr. Aaron Mazzeo for being in the thesis committee.

I would also like to thank all my friends who made the academic life at Rutgers University very enjoyable. I am also thankful to Mr. Paul Pickard and Mr. John Petrowski for teaching me the valuable machine shop skills and etiquettes.

Lastly, I would like to thank my parents and my sister for their mental support to complete my education in the United States. No acknowledgement is complete without thanking them for their support.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

1.1.1 UV-C Decontamination and Sterilization Application

It has been long established that UV-C light has application for decontamination in settings like lab, hospitals and hotels to remove disease carrying airborne microbial organisms. Currently, such systems , as shown in figure 1.1 , has a tower type structure containing the UV-C lights [1]. These towers are human operated and monitored continuously. For efficiency , they can be remote operated too. But with the current progress in the robotic research , specially in indoor environments , this task can be done in an autonomous manner. Automation of this task will assist the personnel and the organization in achieving higher operational efficiency in large environments , specifically hotels. Upgrading the current systems to make them autonomous broadly required completion of two tasks :

• To upgrade the current wall charged system to battery operated for unconstrained navigation

• To make the UV-C degermination application completely autonomous which re-

[1] http://www.clordisys.com/torch.php

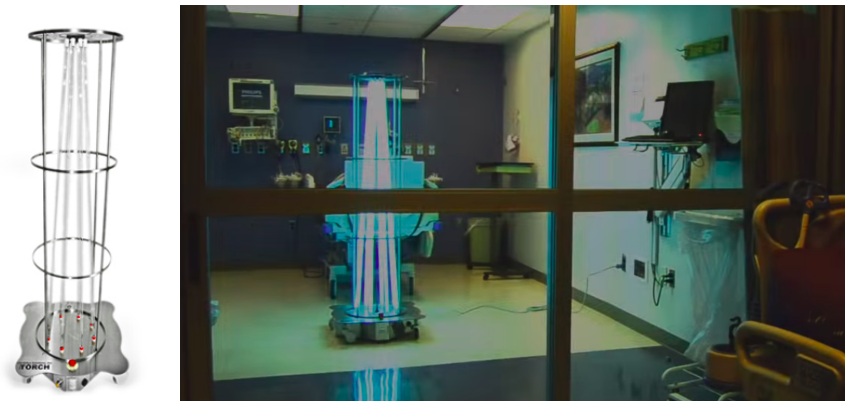


Figure 1.1: Current systems used for UVC degermination , (right) usage in hospital environment

quires minimal human intervention.

Here the autonomy means, that the robot should be able to map the surrounding environment by itself and then prompt the user to input goal locations for UV-C treatment and then navigate to those points autonomously while avoiding obstacles on the way.

1.1.2 Approach

The navigation task can be broken down in four sub tasks : Mapping and Localization, Autonomous Exploration, Obstacle Avoidance and Path Planning. The most basic task is the represenation of the surroundings using SLAM , building on which the other three modules were created. The decision for selecting specific methods for each task is discussed in the next section.

1.2 Literature Review

1.2.1 Robot Mapping and Simulataneous Localization & Mapping (SLAM)

Robot Mapping research is broadly classified by two approaches : Metric and Topological [9]. One of the metric approach is Occupancy grid mapping [10],[11],[12] , which is a discrete representation of the map using finely spaced grid. The topological approach focuses on representing maps by connected objects with the connecting edges giving the information on how to reach from one location to other [13],[14]. Majority of the research in the robot mapping has been devoted to the probabilistic approach [4], specifically in solving the problem of simultaneous localization and mapping . The SLAM algorithms are broadly based on Extended Kalman Filter (EKF) SLAM[17], Graph SLAM or Fast SLAM[16] (based on particle filter [3]). In EKF SLAM , the map is stored as a vector containing either the occupancy probability values in Occupancy Grid methods or as landmark states in feature based SLAM. The map is stochastic and is created by predicting (motion model) and updating (measurement model) using EKF algorithm. All the variables are Gaussian variables (unimodel distribution). Graph SLAM consists of creating pose-graph structures where nodes of the graph are the poses and the connecting edges represent the sensor readings which constraints the two nodes. The problem of Graph SLAM is to find the set of node configurations which has the least error with respects to the measurements. FastSLAM uses Rao-Blackwellized particle filter to estimate robot poses and each particle uses EKF to maintain the state of observed landmark. Since each particle maintains a set of map, the memory requirement is too high for such filters. Such non-parametric filters have the advantage of not being constrained to unimodel distribution for its variables and hence they are better at representing non-linear systems compared to EKF. The advantage of EKF SLAM is that it is computationally faster

and for smaller environments the accuracy of EKF SLAM does not suffer as much as for larger environments. Some other extensions also represent the maps using the raw sensor measurements [15],[2]. For this work , EKF based SLAM called Hector SLAM is used for creating maps.

1.2.2 Obstacle Avoidance

There are various online and offline obstacle avoidance algorithms in literature and in practice. Algorithms like Bug algorithms[18], Virtual Potential Field methods [19],[20], Vector Field Histogram method [21],[22],[23] & bubble band technique [24] are some of the well known algorithms for real time obstacle avoidance. Bug Algorithms are suggested when used with ultrasound sensor , however the path planned is usually not optimal and prone to high sensor noise. Virtual Potential field methods are easier to implement and are more intuitive but can prove to be very computationally intensive in narrow passages. Vector field histograms are more robust when using with occupancy grids since they consider sensor noise and kinematics, but they are computationally more intense. In this work , a combination of offline and online obstacle avoidance method is proposed. The offline version inflates all the obstacles in the map. This acts as the input to the path planner. The online version checks for obstacles on the intended path of the robot using depth sensors. The proposed method not only outputs optimal path, but it also has a wider field of view compared to a single 2D Laser range sensor. Due to need of frequent re-planning of path in case of detection of obstacles, this method demands use of computationally faster path planning methods.

1.2.3 Path Planning

Path Planning methods are broadly based classified in to two types : Deterministic (or Metric) & Probabilistic. The Metric planner are useful on Grid based map repre-

sentation , specially in 2D navigation problems. Some of the widely used algorithms are Dijkstra's Algorithm[25] , A* Algorithm [26] & Wavefront Planning [27]. Dijkstra and A* algorithm are graph based methods. Dijkstra's algorithm finds the shortest distance from source to goal location. It picks the unvisited nodes with the minimum distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. A* algorithm is similar , however it uses heuristic function values to pick the unvisited node with minimum distance. Wavefront planner explores all the cell in the grid map and assigns the distance from the start node as it visits them. Once the goal location is reached, it calculates the path by back tracking through the adjacent cells. Other methods are probabilistic methods like Rapidly-Exploring Random Trees (RRT) [28] & Probabilistic Roadmap (PRM) [29]. RRT creates a tree by exploring in random direction from the start location until the goal location is reached. In PRM , points are sampled randomly in the free configuration space and they are connected to the nearest location in the graph of nodes created so far until it reaches goal location. The probabilistic methods are usually the only effective methods in higher configuration spaces but they take higher computational time. Efficient Extensions for RRTs with lower computation times have also been suggested [31],[30], but the computation time is still higher for 2D spaces compared to the metric methods. Other methods are geometric methods for path planning like Visibility Cell Decomposition , Maximum Clearence Roadmaps and Shortest Path Roadmaps. In these methods the obstacles are approximated as polygons. [5] discusses in details about these methods. This work uses the Dijkstra's & A* search algorithm with orientation compensation with effort to reduce the path planning distance and time. Chapter 4 discusses the computation time of these algorithms which justifies their use for the real time path planning.

1.2.4 Autonomous Exploration

The research on exploring the indoor environment is quite rich. Various techniques have been discussed for detecting unexplored areas using frontiers [1][6], bubble-based exploration [32] or coverage maps [33]. Yamamuchi exploration technique proves to be the most useful with occupancy grids since the other techniques often require a change of the map representation for exploration. Yamamuchi exploration technique relies on the edge detection techqniques used in the image processing. More details on autonomous exploration are discussed in Section 3.7 of Chapter 3.

Chapter 2

System Description

This chapter dives in to a detailed description of the mechanical , embedded and software components of the system. It is divided in to four sub sections :

1. Mechanical Structure
2. Electronic Components
3. Embedded System
4. Software

Some part of the software system also takes help of the open source software packages developed by the Robotics Community. A short summary of that package is also described at the end of the chapter.

2.1 Mechanical Structure

The torch robot is a four wheeled skid steered drive robot with each wheel driven by a motor. The motors are 12V DC brushed motors with PWM input for speed control. Figure 2.1 shows the snapshot of the mechanical construction of the robot.
The steel shell of the robot houses the battery, electronics, torch ballasts and on

board computers.The combination of all the components including the steel shell weighs around 30 pounds. The motors selected have a high Gear ratio (188.6:1) [1]. Hence, they are capable of moving the heavy robot at required speed of 0.5 ft/s without consuming high current.The motors have 12 counts per revolution (CPR) quadrature encoders which were used to provide estimate of the robot pose. With the high gear ratio these encoders can provide an angular resolution of 0.1^o. Chapter 5 discusses briefly about the use of encoders for EKF based robot localization as a scope for improvement.

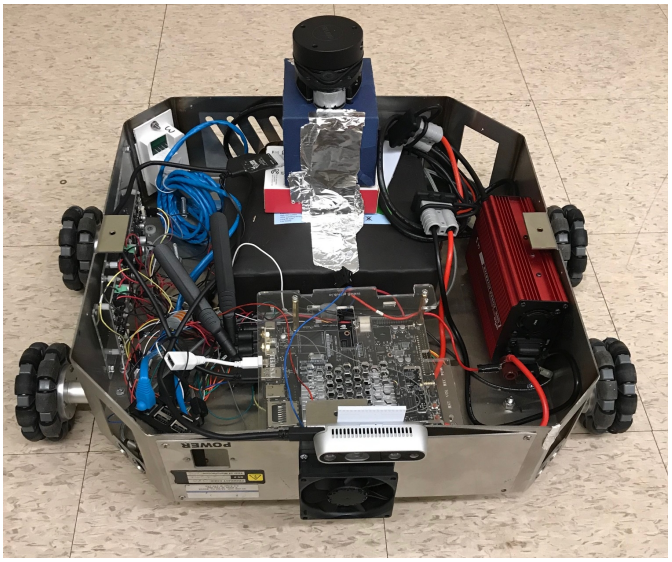


Figure 2.1: Torch Robot

2.2 Electronic components

The electronics of the robot consists of the power supply system, torch circuitry , LiDAR sensor and the motor control boards. The robot houses a 12V 50Ah battery which provides power to the motors , embedded systems (which powers the LiDAR sensor) and the torch. The torches are powered through a DC-AC converter. The

[1]Motor details : https:/www.servocity.com45-rpm-hd-premium-planetary-gear-motor-w-encoder

LiDAR sensor used is manufactured by RPLIDAR (by Slamtec [2]). The LiDAR is Omnidirectional with a 5.5Hz sample frequency and a range of 12 meters. This provides additional mobility to the robot to move in both forward and backward without turning. The LiDAR data is used by the SLAM module which also provides an estimate of the robot's location.

For detecting obstacles, a depth sensing Intel Realsense camera is also used[3]. The camera focuses on the nearby areas for any obstacles in its field of view and updates the map temporarily for the path planner to avoid obstacles dynamically.

The motor control boards have a PWM interface for motor control with a current capacity in the normal motor operating range and hence do not restrict the torque supplied by the motors. The PWM signal is provided by the GPIO pins on the Raspberry pi. The Raspberry pi pins also read the encoder values. Figure 2.2 shows the schematic of the electronic circuit.

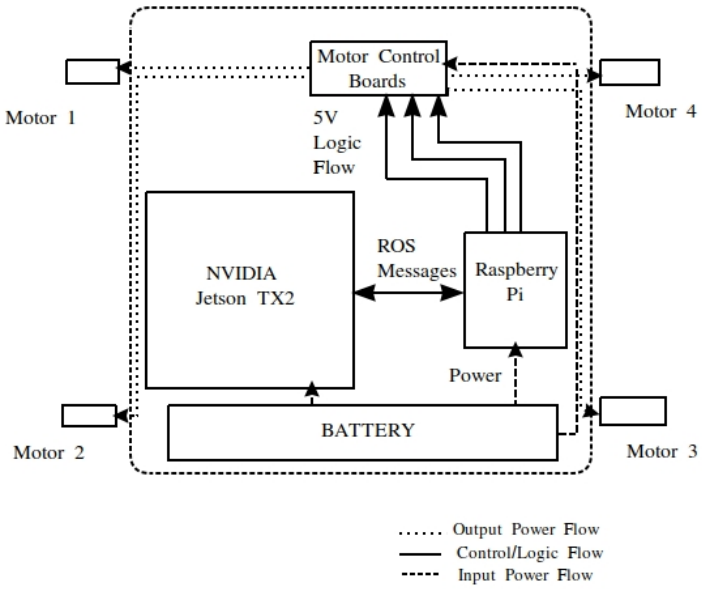


Figure 2.2: Embedded System

[2]http://www.slamtec.com/en/lidar/a1

[3]https://realsense.intel.com/stereo/

2.3 Embedded System

The robot comprises of two onboard embedded systems, NVIDIA Jetson TX2 and Raspberry Pi. However, Raspberry Pi is just for motor control and reading encoder values. Cheaper alternatives exist but Raspberry Pi has robust communication with other computers and it has better ROS (Robot operating System) support, making it a favorable option. Jetson TX2[4] comes with a 6 cores CPU and 256 CUDA cores GPU, providing enough resources to cater to the computation for mapping and motion planning. Jetson TX2 also interfaces with the LiDAR sensor and the Raspberry Pi for sending actuator signals.

2.4 Software

2.4.1 Robot Operating System (ROS)

The official definition of ROS[5] is :

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

While there are multiple software platforms available for Robotics development, ROS is an open source and has a larger support community which makes it easier to develop Robots. Due to its popularity in the robotics community, a wide variety of software packages are available for sensors and actuators, which makes the hardware

[4]https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/
[5]http://wiki.ros.org/ROS/Introduction

interfacing easier and makes the software independent of the embedded system. It also supports a variety of programming languages for accomplishing specific tasks. For the project , C++ and CUDA were used for developing the navigation module of the torch robot.

2.4.2 Compute Unified Device Architecture (CUDA)

CUDA[6] is parallel computing platform and application platform interface (API) developed by NVIDIA for its GPUs. It allows direct access to the parallel elements of the Jetson TX2 board and it is syntactically quite similar to C language, making parallel programming easier.

One of the primary reasons to use GPU for computation is to process a very high number of map grid cells that can be processed at once for obstacle inflation and autonomous exploration. Thus the ratio of computation to memory transfer is higher, justifying the use of GPU. Chapter 3 discusses the usage of GPU in obstacle avoidance and autonomous exploration module.

2.5 Hector SLAM

As discussed at the beginning of the chapter, an open source SLAM software package was used for the development of this robot. This open source, called Hector SLAM [2] was developed at TU Darmstadt in Germany for USAR (Urban Search and Rescue) operations. The difference compared to other well known open source SLAM packages like Gmapping[3] is that this package does not required the odometry of the vehicle since it uses the high speed scan rate of the LiDAR. Chapter 3 dives into a brief discussion about Simultaneous Localization & Mapping (SLAM).

[6]https://en.wikipedia.org/wiki/CUDA

Chapter 3

Navigation System

3.1 Light Detection and Ranging (LiDAR) Sensor

LiDAR is surverying method to represent the objects in the sensor's range by measuring the differences in the time and wavelength of the reflected laser pulse from the object. The LiDAR sensor emits a low power laser (500-600nm wavelength) and calculates the time of return of that pulse. The torch robot uses a RpLidar , which is a 2D Laser range scanner. Being omnidirectional , it outputs the co-ordinates of all the points within 12m range from the location of the robot. In ROS, the laser scan readings are represented by a polar array of the ranges with indexes representing the angles. This array is usually converted to a Point cloud to use it for SLAM. Point cloud is a set of data points in cartesian space. In this case, it is obtained from the LiDAR's location. For a 2D laser range scanner, the data points will be the x-y co-ordinates with LiDAR as the origin.

Figure 3.2 shows the LiDAR output in ROS visualization utility for a scene created in Gazebo simulator.

Figure 3.1: RpLIDAR Sensor

3.2 Occupancy Grid Maps

Occupancy grid maps are the discretized representation of the real world. In this representation the surrounding is represented by a grid of fixed sized cells and each cell is either occupied, free or unknown. The occupancy is represented by probability (0,1). For memory consideration, ROS represents occupancy grid maps with a occupancy probability (0,100). In ROS , the occupancy grid data is a one dimensional array of signed characters. The map info also consists of the resolution, width and height of the map. If the map width is w , then the element (row_num, col_num) in two dimensional grid structure would correspond to $data\_index = col\_num * w + row$.

For each cell in map(m) the probability $p(m_i)$ is given as :

$$p(m_i) = \begin{cases} 100, & occupied \\ 0, & free \\ -1, & unknown \end{cases} \tag{3.1}$$

Figure 3.3 shows a simple representation of an Occupancy grid. Each grid cell in

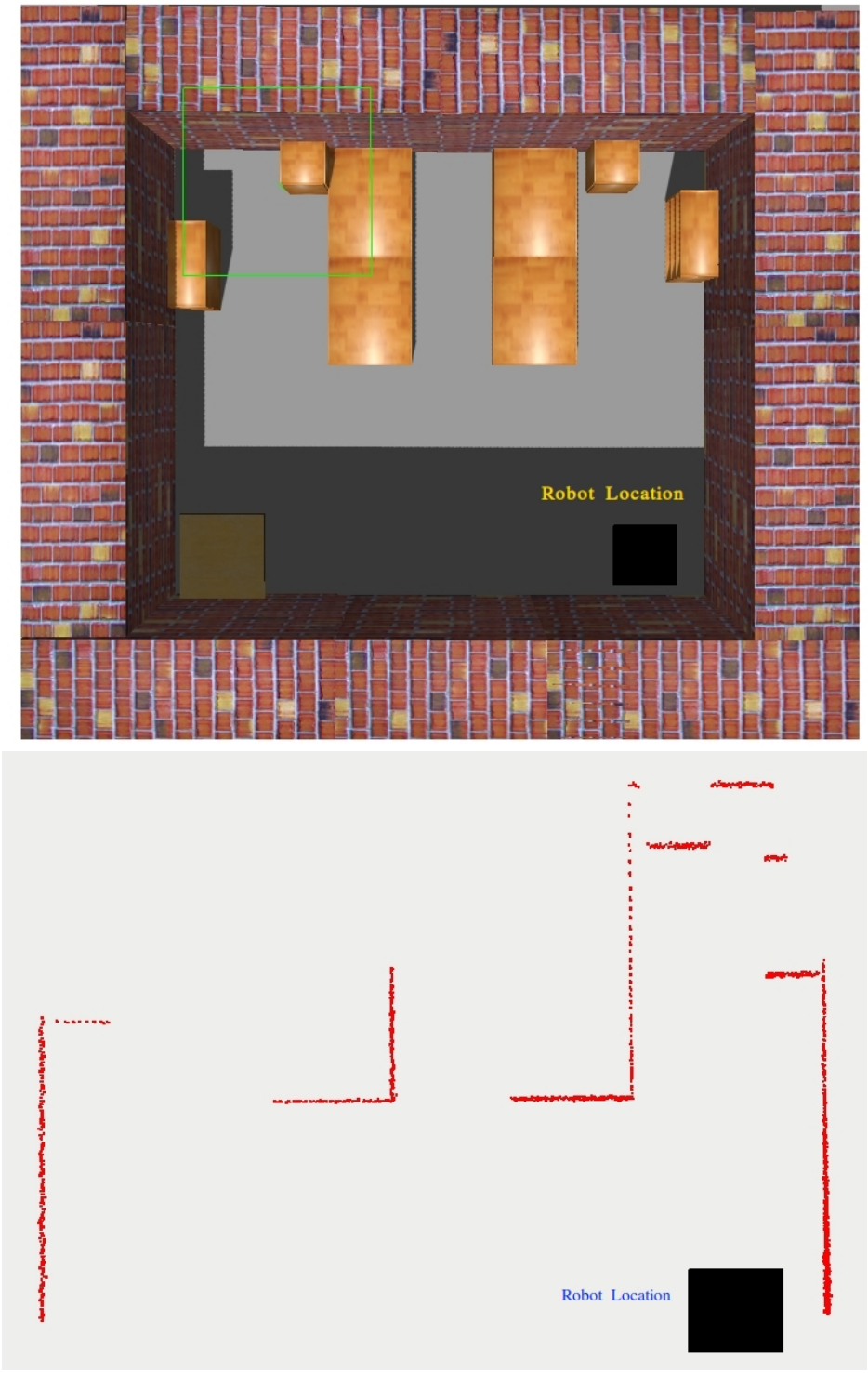


Figure 3.2: LiDAR Output

the grid has a fixed size , which defines its resolution. For eg., the map of a room of size 20m x 20m with each grid of 0.05m resolution, will have 160,000 grid cells. This

makes processing the grids for path planning and obstacle avoidance computationally expensive and makes the map rigid , but it does not require any feature detection to create maps. It simply maps the point cloud at the location of corresponding grid cells and fills it with the occupancy probability. The process of constructing such maps using various mapping algorithms is Occupancy Grid Mapping. Mathematically , it is a Bayes filter with posterior probability given as the product over the individual cells :

$$p(m|z_{1:t}x_{1:t}) = \prod_i p(m_i|z_{1:t}x_{1:t}) \tag{3.2}$$

where, $z_{1:t}$ is the sensor data, $x_{1:t}$ is the robot pose data. The Algorithm 1 [4] briefly explains the process of constructing Occupancy Grids.

Hector Mapping, used in the torch robot is a SLAM algorithm used for generating Occupancy Grid Maps. Figure 3.4 shows the Occupancy Grid generated using a LiDAR for a typical surrounding .

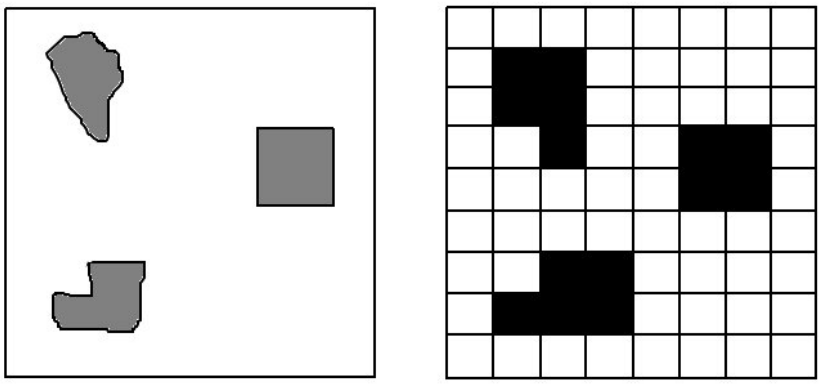


Figure 3.3: Simple Example of an Occupancy Grid , left figure is the Real world with objects in it, right figure is the corresponding Occupancy Grid Representation.

Figure 3.4: Occupancy Grid Output of the Room

Algorithm 1 Occupancy Grid Mapping

procedure OCCUPANCY GRID MAPPING
Require: $l_{t-1,i}$,x_t,z_t,M_i
1: **for** all cells m_i in M_i **do**
2: **if** m_i in the measurement range of z_i **then**
3: $l_{t,i} = l_{t-1,i}$ + inverse_sensor_model$(m_i, x_t, z_t) - l_0$
4: **else**
$l_{t,i} = l_{t-1,i}$
5: **end if**
6: **end for**
7: **return** $\{l_{t,i}\}$

In Algorithm 1, the $l_{t,i}$ is the log odds ratio, which is given as :

$$l_{t,i} = log\frac{p(m_i|z_{1:t}, x_{1:t})}{1 - p(m_i|z_{1:t}, x_{1:t})} \tag{3.3}$$

Also,

$$inverse\_sensor\_model(m_i, x_t, z_t) = log\frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)} \tag{3.4}$$

And,

$$l_0 = log\frac{p(m_i)}{1 - p(m_i)} \tag{3.5}$$

The log odds value varry from $[-\infty, \infty]$ and hence avoid loss of precision in proba-

bility representation when the probabilities value approach 1 or 0.

3.3 Hector SLAM

The torch robot uses the open source Hector SLAM package for simultaneous localization and mapping. Hector SLAM uses the scan matching technique to align the laser scans and provide the pose of the robot that leads to this change. Simply put, it finds out the transformation between the two scans and assumes that the difference is due to the movement of the robot. It does so by optimization of the alignment of the beam endpoints with the map created so far using Gauss Newton Approach[2] .Mathematical derivations of the Hector Scan Matching are duplicated in Appendix A for reference. This approach proves to be accurate most of the time in an indoor environment as long as the LiDAR scan rate is higher enough than the speed of the robot or the range of LiDAR is high enough to capture multiple landmarks in its surrounding.

A brief idea of the Hector SLAM 2D subsystem in presented below :

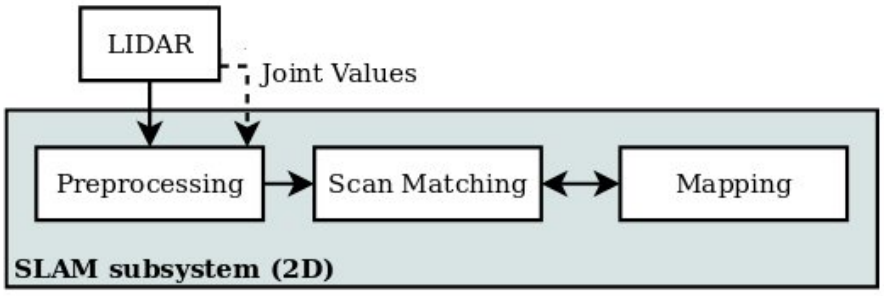


Figure 3.5: Hector SLAM 2D subsystem [Image source : [2]]

3.4 Obstacle Inflation (Costmap)

The Occupancy Map created by the SLAM system consists of grid cells with their respective occupancy probability. Any consequent motion planning will involve accessing these grid cells to establish if that grid cell is safe for robot to be at the location of grid cell. Even though the mapping might produce a grid cell which is free, its distance from the nearest occupied space/grid cell might be less than the robot's physical size. If the path planner outputs a path which passes through such point(s), the robot might collide with the obstacle. This demands a need to check if the grid cell under consideration is safe or not. This can be accomplished by inflating all the occupied grid cells as per the robot's physical dimension. This can prove to be a computationally intense task. While there are open source options [1] available for obstacle inflation using the sensor data , they are optimized to work with static maps. The present work uses the map learnt so far using the SLAM system to inflate the obstacles using parallel programming. This makes the navigation module of the robot more responsive to the map changes for robust path planning. Figure 3.6 shows the idea for obstacle inflation.

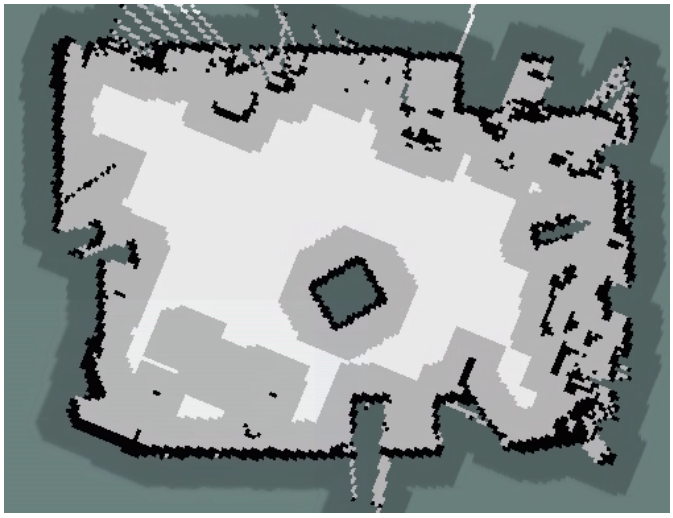


Figure 3.6: Obstacle Inflation example, black solid lines is the map , shaded grey part is the inflated region

[1] http://wiki.ros.org/costmap_2d

The GPU operation can be interpreted as a stencil moving at each grid cells location to look if there are any occupied cells in that stencil's area.The size of that stencil will be as per the robot's physical dimension.

3.5 Obstacle Detection & Integration

The RpLiDAR discussed in section 3.1 is a single channel 2D LiDAR , meaning it outputs scans only at the sensor's height. A more reliable method would be to use another sensor to compliment the map generated by the LiDAR. This section dives into more detail in accomplishing this task using the 3D point cloud generated by a depth sensing stereo camera to selectively update the occupancy grid in the vicinity of robot. The obstacle data obtained using this method is projected to the plane of the LiDAR sensor and is used to update the occupancy grid which is fed as input for obstacle inflation.

The stereo camera Intel Realsense D435 was used for the experiments for selective updating of the Occupancy grid. This camera operates at 30fps and every point cloud output has upto 16,000 points with a resolution as low as 0.01m. Hence a voxel filter is employed to downsample this point cloud with a resolution of 0.1m.

3.5.1 Voxel Filter

Voxel filter downsamples the point cloud data by calculating the centroid/spatial average of the point in a subspace. It does so by finding the nearest neighbors in the subspace and calculates the centroid of all the points. For eg. if we want to downsample the point cloud to represent 0.1m radius spheres, all the the points in that spherical subspace will will be replaced by a single point representing a 0.1m sphere. Figure 3.7 shows an some example of this filter's result.

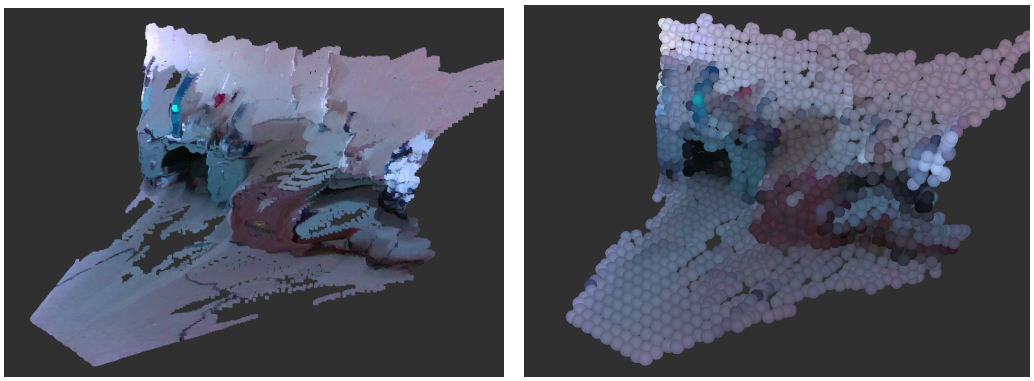


Figure 3.7: Point Cloud Comparision , left side figure is the normal point cloud and right side is the filtered cloud

Algorithm 2 Selective Obstacle Update

procedure ObstacleDetection
Require: pt_cloud,d_{max}
1: obstacle_indices = $\emptyset$
2: filtered_grid = VoxelFilter(pt_cloud)
3: **for** p_i in filtered_cloud **do**
4: **if** $p_i \leq d_{max}$ and $p_i > z_{ground}$ **then**
5: $p(m_i)$ = **map_index**(p_i)
6: obstacle_indices[i] = $p(m_i)$
7: **end if**
8: **end for**
9: **return** obstacle_indices

3.5.2 Selective Updating of the Occupancy Grid

Once the downsampling of the data is complete, all the points in the vicinity of the robot are projected onto the LiDAR plane and are mapped to repsepctive grid cell in the occupancy grid. This approach offers two advantages :

1. Due to downsampling, the update of the occupancy grid is computationally less intense.

2. Due to increased field of view the path planning is more robust in avoiding static and dynamic obstacles.

Algorithm 2 shows the brief idea of integrating obstacle detected from point cloud.

3.6 Path Planning

Path planning is the task of creating plans for the robot to manipulate it current state to the desired state. As per [5] , the basic ingredients of planning are :

• **State space** - a list of all the feasible states the robot can occupy

• **Time** - the time over which all the actions are applied or the succession in which the actions are applied.

• **Actions** - The set of input or controls, which manipulate the state of the robot.

• **Initial and Goal states** - The starting state and the state aimed to be reached using the plan

• **Criterion** - The criteria which the plan must be accomplishing. It can be to make the plan optimal w.r.t. time or energy consumed or distance travelled or it can simply be to find out if the plan is feasible or not.

3.6.1 Planning Problem Formulation

Since we chose to represent the surrounding world as an Occupancy Grid, which is a discrete representation, a planning problem formulation needs to be done for a discrete space.

In terms of the torch robot, the planning problem is to seek a plan from the initial robot state x_I to goal state x_G. Since the robot yaw angle at particular points in the planned path is not a specific requirement, the initial state, the goal state and the intermediate states are just position vectors :

$$x_k = [x \;\; y]^T \tag{3.6}$$

Thus, the Configuration space C for planning purpose is 2D.

The state space , X is a list of all the feasible states. In the case of an Occupancy

Grid, X would be a list of all the grid cells which are not occupied at that instant.

The planning problem for the mobile torch robot can be formulated as :

- A set of non empty state space X
- A initial state x_I and the goal state x_G
- A control action $U(x)$ for each $x \epsilon X$.
- A state transition function $x^{'} = f(x, u)$, which manipulates the state of the robot to $x^{'}$ from x.

A convenient way of representing this problem is using Graphs. A Graph G is a set of vertices (V) and edges (E) between the vertices. In terms of planning, the vertices of the graph are the grid cells in the map. The edge between them denotes that they are connected and the weight of the edges denotes the cost of the control action to go from one node to the other ($f(x, u)$). A graph can be a directed graph or an undirected graph. In Occupancy grids, it can be easily interpreted that a undirected graph can be used since the grid cells can be travelled back and forth between themselves.

3.6.2 Graph Edge Weight Calculations

Instead of considering the edge from current cell to adjacent grid cells to have same cost, a robot pose based scheme was used to give edgeweights to the edge from current to neighbouring cell/nodes in the graph.

The Wavefront Algorithm [27] assigns uniform cost to each of the neighouring cells. But for this work, the weight of the edges is defined based on the start pose of the robot and the distance from the start.

Given a grid cell under consideration x_k and let x_{kadj} be the adjacent cell of x_k. Let the distance from the robot initial location x_I to x_{kadj} be d. The angle(θ) of the line joining x_k and x_{kadj} is a multiple of $\pi/4$ as shown in the fig 3.8. This angle is modified according to the equation

$$\theta_{mod} = \begin{cases} \theta + \phi & d < d_{max} \\ \theta & d \geq d_{max} \end{cases} \tag{3.7}$$

where, ϕ is the yaw angle of the robot and d_{max} is the maximum value for which the angle has to be modified.

The reason for modifying the angle to shorten the time to travel . For a distance under d_{max} , the angle of the line joining x_k and x_{kadj} is modified to align with the yaw angle of the robot. This means that the path planner will output a path which prefers the robot to travel straight initially compared to performing the turn operation, which has higher time cost.

The edge cost can now be calculated as :

$$EdgeCost = \begin{cases} \alpha & \theta_{mod} \ \epsilon \ [0, \pi] \\ 1.05\alpha & \theta_{mod} \ \epsilon \ [\frac{\pi}{4}, \frac{3\pi}{4}, \frac{5\pi}{4}, \frac{7\pi}{4}] \end{cases} \tag{3.8}$$

where, α can be an arbitrary constant. The obvious selection would be the resolution of the grid. However to save the memory , this value can be chosen to be 50 to avoid creating floating point numbers for implementation purposes.

Once a Graph is created, shortest path algorithms can be applied to find out the path from the initial state/node to the goal state/node. This work compares two of such well known algorithms, which optimizes the sum of the edgeweights.

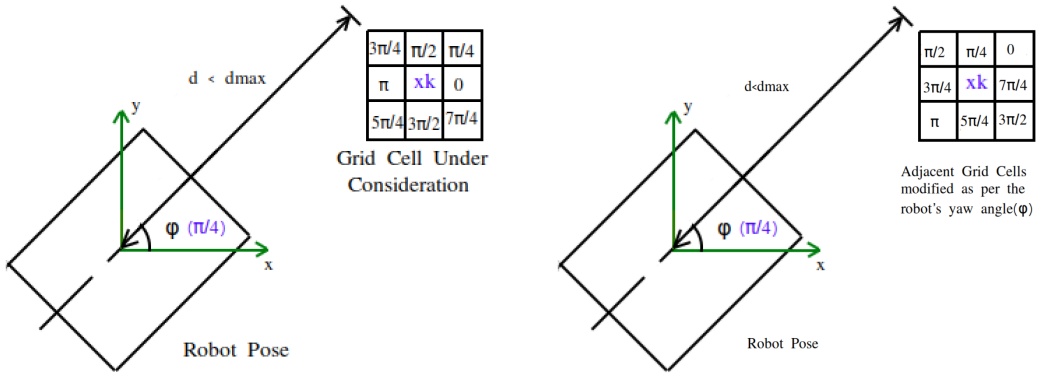


Figure 3.8: Figure shows the effect of the angle modification according to the Robot's pose while building the graph

3.6.3 Shortest-Path Algorithms

Shortest path algorithms find the shortest path between two vertices in the graph. Dijkstra's algorithm is the most well known algorithm. However it finds the shortest path between source vertex to all vertex in the graph until goal vertex is reached. It uniformly goes through all the vertex until the goal vertex is reached. The other variant in A* search, which considers a heuristic function to choose the next vertex while calculating the path. This work compares these two algorithms in terms of Forward Search. Forward search means searching for goal node with initial node as the root node.

These methods are deterministic methods compared to the sampling based methods like RRT and PRM. The complexity of these methods, which will be explored later is dependent on the number of the grid cells in the graph created from the initial node to the goal node. These methods do not suffer from the local minima problem faced while using greedy algorithms like Virtual field methods. The Graph based planning methods do face a disadvantage when the maps are bigger or dimension of the configuration space is higher since that leads to an increase in the number of the nodes in the graph.

Dijkstra's Algorithm

For a given starting node, Dijkstra's Algorithm finds the optimal path to every node in the graph. It optimizes the sum of the edges from the root node x_I to any node x in the graph. The complexity of the Dijkstra's Algorithm is $O(|V|log(|V|) + |E|)$ [34], where V is the number of vertices in the graph and E is the number of edges in the graph.

The step-by-step procedure for Dijkstra's algorithm is :

1) Mark all the nodes in the graph as unvisited

2) Set the start node (x_I) as the current node and assign an arbitrary high value to all other nodes in the graph (infinity).

3) For all the neighbours of the current node, compute the cost-to-go from the current node. Compare this tentative cost with the the current cost-to-go value assigned to the node. If the tenative cost is lower, then replace the current cost with lower cost.

4) Mark this node as visited. If the goal node is visited, stop the algorithm.

5) Make the next unvisited node in the queue as the current node and repeat step 3.

Algorithm 3 gives the general idea of the Dijkstra's Algorithm used for path planning on Occupancy grid.

A* Algorithm

A* algorithm combines the benefit of the informed search like Best first search [37] and Dijkstra's algorithm in searching the path. The cost-to-go value while exploring the nodes is the combination of the cost from the current node to the next node and the heuristic function value for that node. This algorithm does not guarantee optimality since the result depends on the type of heuristic function. The general idea of the algorithm is given as :

Algorithm 3 Dijkstra's Algorithm

procedure DIJKSTRA'S ALGORITHM
Require: *map*,*source_node*,*goal_node*
visited_nodes=$\emptyset$,queue=*source_node*
for all GridCell in map **do**
    dist[GridCell] = INF
    predecessor[GridCell] = UNKNOWN
end for
dist[source_node] = 0
while goal_node not in visited_nodes AND !queue.empty() **do**
    current_node = minDistance(queue)
    for each neighbour i of current_node **do**
        if map[i] not occupied **then**
            CostToGo = dist[current_node] + *EdgeCost*
            if CostToGo < dist[i] **then**
                dist[i] = CostToGo
                predecessor[i] = current_node
            end if
        end if
    end for
    visited[current_node] = true
end while
return predecessor[]
end procedure

1) Mark all the nodes as unvisited

2) Mark the start node as current node

3) Expore all the neighbouring nodes and assign them cost-to-go value from the current node plus the heuristic function value

4) Mark the current node as visited

5) If the goal node is marked as visited, stop the algorithm or else

6) Choose the best node to explore based on the value calculated above and make that node as current node and repeat from step 3 . Algorithm 4 describes the A* algorithm.

Algorithm 4 A* Algorithm

procedure A* ALGORITHM
Require: *map*,*source_node*,*goal_node*
visited_nodes=∅,queue=*source_node*
for all GridCell in map **do**
  dist[GridCell] = INF
  predecessor[GridCell] = UNKNOWN
end for
dist[source_node] = 0
while goal_node not in visited_nodes AND !queue.empty() **do**
  current_node = minDistance(queue)
  for each neighbour i of current_node **do**
    if map[i] not occupied **then**
      CostToGo = dist[current_node] + *EdgeCost* + heuristicValue(current_node)
      if CostToGo < dist[i] **then**
        dist[i] = CostToGo
        predecessor[i] = current_node
      end if
    end if
  end for
  visited[current_node] = true
end while
return predecessor[]
end procedure

3.7 Autonomous Exploration

Autonomous exploration is the problem of searching for areas in the map, which are yet to be explored. In Occupancy grids , the unknown grid cells are assigned values of -1. For exploration purpose, the aim is to find such unknown area which will help in completing the map of the surrounding. For example, for a indoor robot, the aim of the autonomous exploration is to create a complete map of the room. [6] demonstrates the use of Yamamuchi frontiers for autonomous exploration. Yamamuchi [1] demonstrated the use of edge detection techniques used in image processing for autonomous exploration. For this work , we use the sobel edge detection [7] method using parallel processing for autonomous exploration.

3.7.1 Sobel Edge Detection

In case of maps, edges are the location when a change occurs. For occupancy grids this happens at the boundary where on one side the cells are either free ($p(m_i) = 0$) and other side they are unknown ($p(m_i) = -1$). Figure 3.9 shows this situation. The gradient of these values will give an idea about the edge between free space and unknown space.

In case of the occupancy grid, the derivative will be like finite differences along both the axes as shown below :

$$map_{ix} = f(x+1) - f(x) \tag{3.9}$$

$$map_{iy} = f(y+1) - f(y) \tag{3.10}$$

where, map_i is the grid cell under consideration.

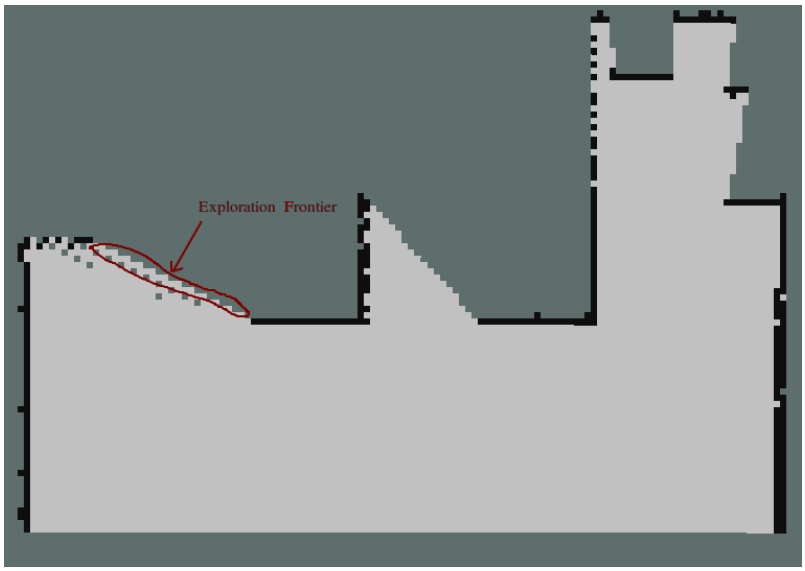


Figure 3.9: Example of Exploration Frontier

Better approximation of the derivative would be :

$$m_{ix} = 2f(x+1,y)+f(x+1,y+1)+f(x+1,y-1)-2f(x-1,y)-f(x-1,y+1)-f(x-1,y-1) \tag{3.11}$$

$$m_{iy} = 2f(x,y+1)+f(x-1,y+1)+f(x+1,y+1)-2f(x,y-1)-f(x-1,y-1)-f(x+1,y-1) \tag{3.12}$$

The equations can be written as 9x9 matrix type operator , called kernel.

$$s_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{3.13}$$

$$s_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3.14}$$

Kernel 3.13 & 3.14 are called the sobel operators. When we do a convolution of an occupancy grid matrix using this kernel, we effectively calculate the gradients

of the values of the grid in both directions. Each gridcell in the convoluted map will have the value as per the equation:

$$M = \sqrt{s_x^2 + s_y^2} \tag{3.15}$$

Figure 3.10 shows some examples of the value of M for various cases in the Occupancy Grid. These calculations gives an idea about the numerical value of M which indicates an unknown frontier. Figure 3.11 shows the result of convolution of the whole map. Calculation of different scenarios in a map, shows that the frontier between a free area and an unknown area would lie in the range M $\epsilon[1, 6]$.

Using this information , an algorithm to find out the unexplored regions in the map was devised. Algorithm 5 explains this idea.

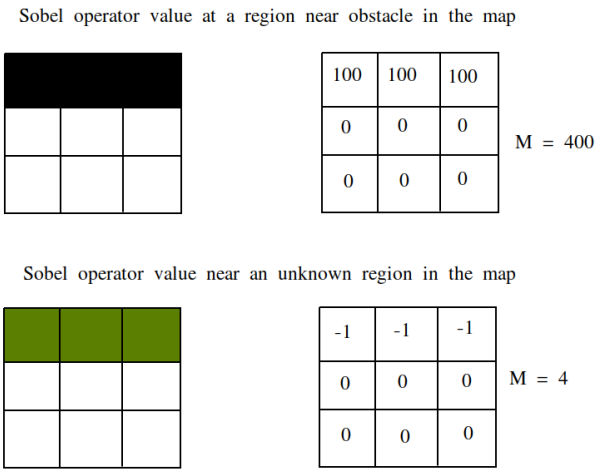


Figure 3.10: Calculation of Sobel operator in different regions in map

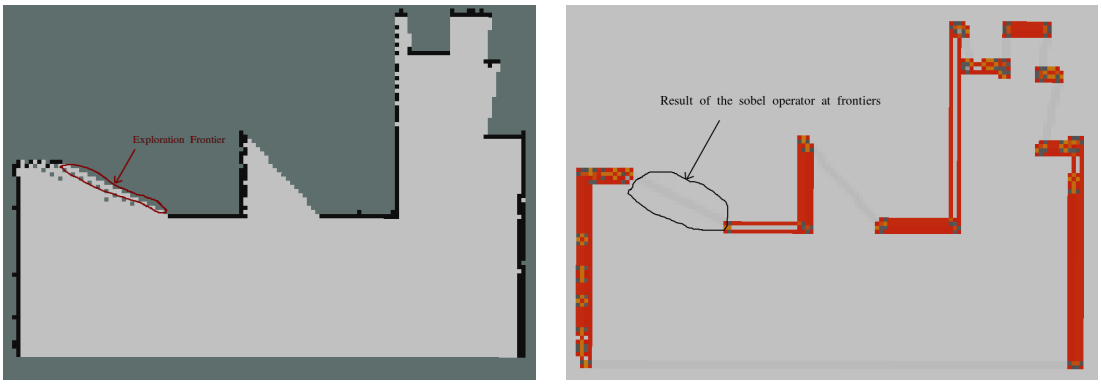


Figure 3.11: (top)original map, (bottom)Result of the sobel operator on the whole map, gray areas shows transition into unknown area

3.7.2 GPU Acceleration

Convolution of the map using a sobel operator makes a good case to apply data parallelism. Performing the convolution computation on the GPU can lead to decreased runtime for finding frontiers in the map. Figure 3.12 shows the GPU accelerated model for performing sobel operator.

3.7.3 Frontier Detection

Once the sobel operation on the map is completed, an array of possible frontiers is made. For each frontier , the path planner determines whether a path exists to that point in the map. This operation is continued until a feasible path is obtained.

The frontier arrays obtained from the Algorithm 5 will be the input for the path planner. Path planner will operation will stop once it can successfully run on any point in the frontier array and outputs the resulting path.

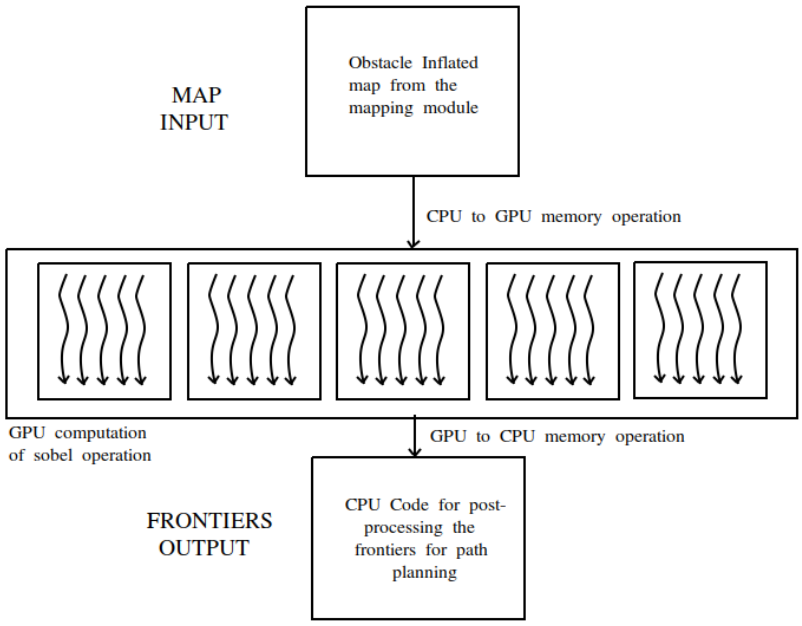


Figure 3.12: GPU acceleration for froniter detection

Algorithm 5 Autonomous Exploration

procedure AUTONOMOUS EXPLORATION
Require: *map*, nearFrontierArray = ∅, farFrontierArray = ∅
for each GridCell u in map **do**
  dist[u] = INF
end for
for each Gridcell u in map **do**
  M = sobelOperation(Gridcell)
  if M ϵ [1,6] **then**
    dist[u] = euclideanDistance(robotPosition,u)
  end if
end for
for each point k in dist[] **do**
  if dist[k] < nearFrontierDistance **then**
    nearFrontierArray.add(k) dist[k] > nearFrontierDistance & dist[k] < farFrontierDistance
    farFrontierArray.add(k)
  end if
end for
return farFrontierArray,nearFronitierArray
end procedure

3.8 Kinematics

Understanding the kinematics is vital to understand behaviour of the robot under various types of input and also to create a controller for the robot to achieve a speicifc motion. For mobile robots, the understanding of kinematics is also important for position estimation of the robot [4],[8]. This section develops the kinematic model of the robot for creating a controller to follow the path created by the path planner.

3.8.1 Robot Pose

The torch robot is a four wheel skid steered drive robot and hence the pose of the robot will be a 3 dimensional vector.

$$\xi_I = [x\ y\ \theta\ ]^T \tag{3.16}$$

Figure 3.13 shows the description of robot pose w.r.t. local and global reference frame. Here, $[X_G, Y_G]$ is the global inertial reference frame and $[X_R, Y_R]$ is the robot's local frame. The robot local reference frame is defined by two axes with the origin at point P on the chassis of the robot. The position of this local reference frame is defined by the co-ordinates x and y. The orientation between them is defined by the angle θ. The rotation matrix for a robot pose ξ_R in local frame to ξ_G in the global frame is given as :

$$R(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

Hence , the transform between the poses ξ_R to ξ_G is given as :

$$\xi_I = R(\theta)\xi_R \tag{3.18}$$

Since the robot is a four wheeled drive and the speed of the front and the rear wheel on each side would be the same the robot motion would be like a skid steer. The kinematics would be similar to a 2 wheel differential drive robot but not excatly the same due to violation of sliding constraints.

Let the robot position be P. If the left wheel rotation speed is $\dot{\phi}_1$ and the right wheel rotation is $\dot{\phi}_2$, then the robot position in its local frame will be given as :

$$\dot{\xi_R} = \begin{bmatrix} \frac{d\dot{\phi_2}}{2} + \frac{d\dot{\phi_2}}{2} \\ 0 \\ \frac{d\dot{\phi_2}}{2l} + \frac{-d\dot{\phi_2}}{2l} \end{bmatrix} \tag{3.19}$$

Here,

d = diameter of the wheel

l = wheelbase of the robot (distance between the right and left wheels)

The robot pose in the global frame would be given by :

$$\dot{\xi}_I = R(\theta) \begin{bmatrix} \frac{d\dot{\phi_2}}{2} + \frac{d\dot{\phi_2}}{2} \\ 0 \\ \frac{d\dot{\phi_2}}{2l} + \frac{-d\dot{\phi_2}}{2l} \end{bmatrix} \tag{3.20}$$

3.8.2 System Description

The state transition equation for the four wheeled torch robot (skid steer geometry) is given as :

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.21}$$

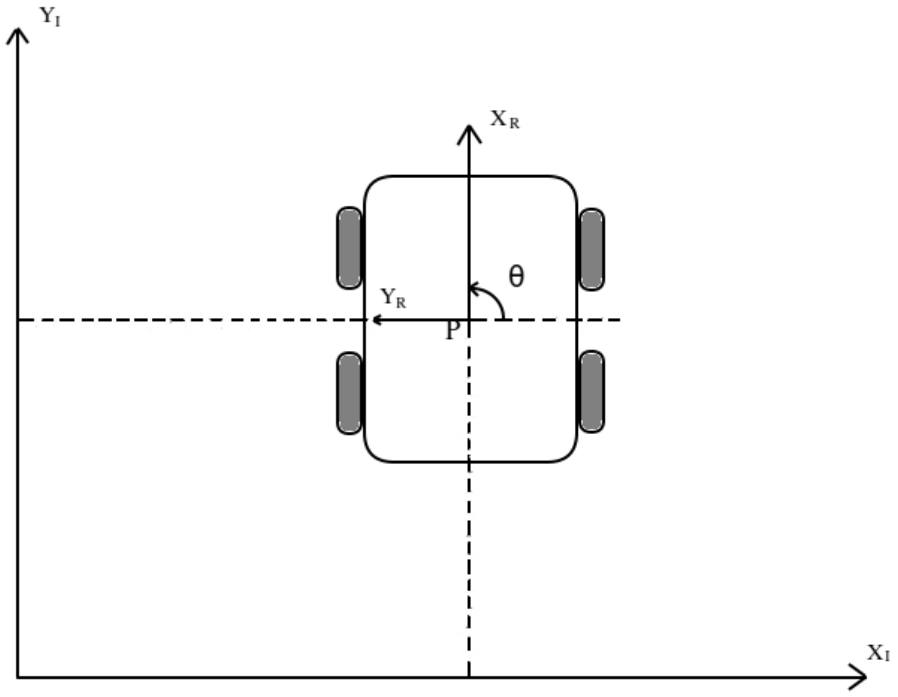


Figure 3.13: Robot position description in Local and Global reference frame

Equation [3.21] can be alternatively represented by a *driftless control affine system*

$$\dot{\mathbf{x}} = f(x)v + g(x)\omega \tag{3.22}$$

The control system for the robot to follow a certain trajectory will be studied in the next section w.r.t. the above equation with linear and angular velocities as input. The skid steered geometry of the has close similarities to differential drive robot w.r.t the inputs. However , the lateral slip constraint is violated since all the wheels do not share the same horizaontal axis. This makes the estimation of the robot location using wheel encoder very difficult leading to more reliance on localization using exteroceptive sensors like LiDAR or visual odometry. But skid steered geometry

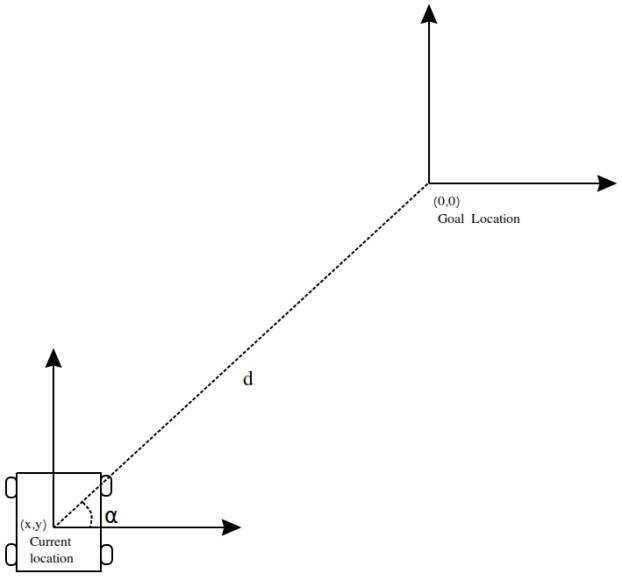


Figure 3.14: Robot pose error for followring a point

has certain advatages due to added torque of additional motors which makes the motor selection easier.

3.9 Feedback Control

Controlling the motors of the robot to follow the trajectory can simply be broken down in to following points on the trajectory. As seen in the previous sections, two control inputs to the mobile robot system are linear velocity(v) and angular velocity(ω). Therefore, a controller to follow a point is developed using methods from [8].

3.9.1 Following a Point

The error modelling for the robot is more intuitive in terms of polar co-ordinates as shown in the figure 3.14.

Here ,

$$d = \sqrt{\Delta x^2 + \Delta y^2} = \sqrt{(x-0)^2 + (y-0)^2} = \sqrt{x^2 + y^2} \tag{3.23}$$

is the distance from the robot position to the goal position , which is the origin. Also the angle difference between the robot orientation and the line joining the current robot position and the goal point can be expressed as :

$$\alpha = -\theta + atan2(\Delta y, \Delta x) = -\theta + atan2(y, x) \tag{3.24}$$

$$\beta = -\theta - \alpha \tag{3.25}$$

The goal of the controller is to provide input to the system to reach the goal point and hence making the error terms tend to zero. The relation between these error terms and the robot velocity vector is :

$$\begin{bmatrix} \dot{d} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -cos\alpha & 0 \\ \frac{sin\alpha}{d} & -1 \\ -\frac{sin\alpha}{d} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.26}$$

β is the desired orientation of the robot. If the goal is just to reach the desired points irrespective of any pose, the error formulation to reach a point is :

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} -cos\alpha & 0 \\ \frac{sin\alpha}{\rho} & -1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.27}$$

Let the Control law be :

$$v = k_\rho \rho \tag{3.28}$$

$$\omega = k_\alpha \alpha \tag{3.29}$$

Substituting these input in the equation 3.27 , we get the resulting closed loop system as :

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} -k_\rho \rho cos\alpha \\ k_\rho sin\alpha - k_\alpha \alpha \end{bmatrix} \tag{3.30}$$

The equilibrium of the closed loop system is $(\rho, \alpha) = (0, 0)$. The system can be linearized at equilibrium position using $sin(\alpha) = \alpha$ and $cos(\alpha) = 1$.

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} -k_\rho & 0 \\ 0 & k_\rho - k_\alpha \end{bmatrix} \begin{bmatrix} \rho \\ \alpha \end{bmatrix} \tag{3.31}$$

The characteristic polynomial of the system is :

$$\lambda^2 + \lambda k_\rho + k_\rho^2 - k_\alpha k_\rho = 0 \tag{3.32}$$

For the eigenvalues (λ) to be negative for system stability the following conditions should be satisfied :

$$k_\rho > 0 \qquad \& \qquad k_\alpha > k_\rho \tag{3.33}$$

3.9.2 Following a trajectory

The trajectory generated by a path planner is a list of points. So the idea of the controller to follow a point can be extended to follow the trajectory. An idea of the algorithm for following the trajectory is given in Algorithm 6

Distance e_{min} on line 1 is the allowable error in reaching the end point of the trajectory.

The d_{near} distance in the algorithm above is for selecting the point to follow in the trajectory. The *nearestPoint* function finds out the point in the trajectory which is at least d_{near} distance from the current robot position and discards the points that are at a lower distance than that point.

Line 5-9 performs the angle modifications, which allows the robot to travel in both directions. It adds or subtracts $\frac{\pi}{2}$ so that the angle error stays in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$. It updates the direction , which is the input in the velocity equation on line 12.

Algorithm 6 Trajectory following

procedure TRAJECTORY FOLLOWING

Require: d_{near},k_ρ,k_α, path,current_pose

while distance(robotPose,path.back()) $< e_{min}$ **do**
  point = nearestPoint(path,d_{near})
  distance2goal = euclidDistance(robotPose,path.back())
  $\alpha = -\theta + atan2(\Delta y, \Delta x)$
  if $\alpha > \frac{\pi}{2}$ **then**
    $\alpha = \alpha - \frac{\pi}{2}$
    direction = -1
  else if $\alpha < -\frac{\pi}{2}$ **then**
    $\alpha = \alpha + \frac{\pi}{2}$
    direction = -1
  end if
  v = k_ρ(distance2goal) $\times$ direction
  $\omega = k_\alpha \alpha$
end while

end procedure

Chapter 4

Simulation & Experimental Results

4.1 Simulation Environment

A robotic simulator is used to simulate the physical robot and its behaviour without the use of the hardware and sensors. In mobile robots, various simulators are available which are used to build different scenarios in which the robot will operate. In the present work , Gazebo simulator was used. It is developed by Open Source Robotics Foundation [1]. Gazebo simulator provides a very strong and convenient integration with the ROS middleware and hence proves to be a very good testing environment before deploying the whole system on the physical hardware.

For reliable results a robot similar to the torch robot was built in the simulation and it was tested under a environment similar to its intended use. Figure 4.1 shows the simulation environment.

4.2 Obstacle Inflation

The obstacle inflation operation is done on the GPU for higher speed. The run time on CPU and GPU is also discussed at the end of the section. The CPU used for

[1] https://www.openrobotics.org/

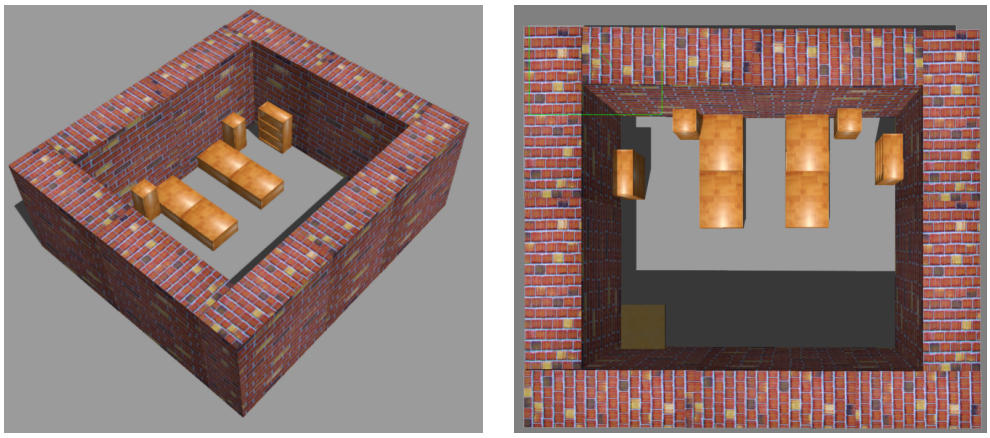


Figure 4.1: Simulation Environment

comparison was the computer equipped with 8th generation Intel i5 processor.

4.2.1 Experimental

The experiments were carried out in the indoor lab environment. The smallest room size in which the experiments were performed was 10m x 8m. The experimental results for the obstacle inflation module are shown in Figure 4.2. The run time for the obstacle inflation module on the NVIDIA Jetson GPU is on average 48ms comapared to 400ms on a Intel i5 CPU.

4.2.2 Obstacle Detection

The results shown in the figures 4.4 & 4.5 demonstrate the effect of using the camera to detect obstacles not in LiDAR's field of view.

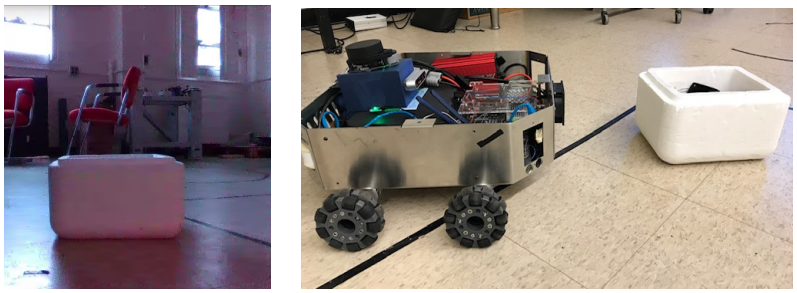


Figure 4.4: Obstacle in front of the Robot (not in LiDAR's field of view)

Figure 4.2: Obstacle Inflation Result (White Spaces is the obstacle free space)

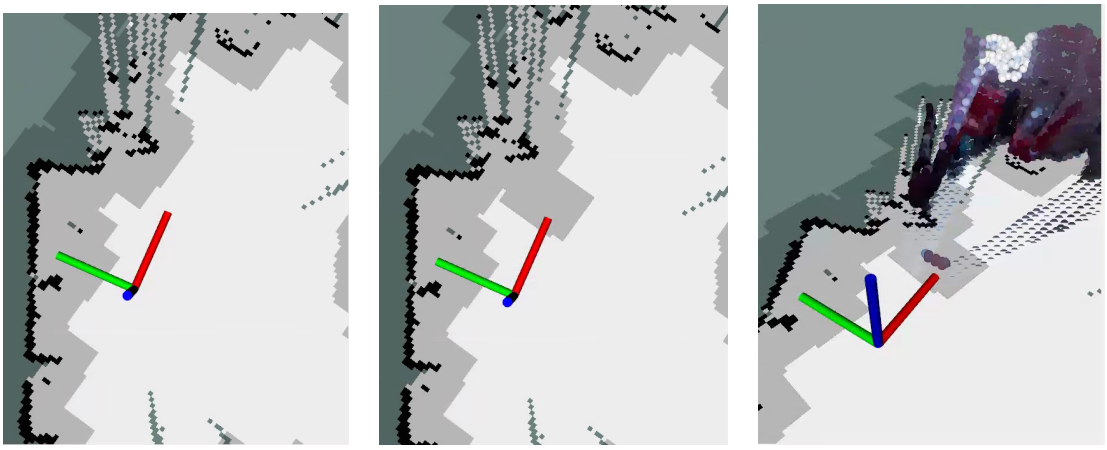


Figure 4.5: Obstacle Detected by the Camera added to the occupancy grid , (topleft) Result without Camera (topright)Result with Camera, (bottom) Voxel Filter imposed on the Occupancy Grid for visualization

4.3 Path Planning

The input to the path planner are the cartesian co-ordinates of the goal provided by the user. For the purpose of testing, the Rviz utility was used which provides click

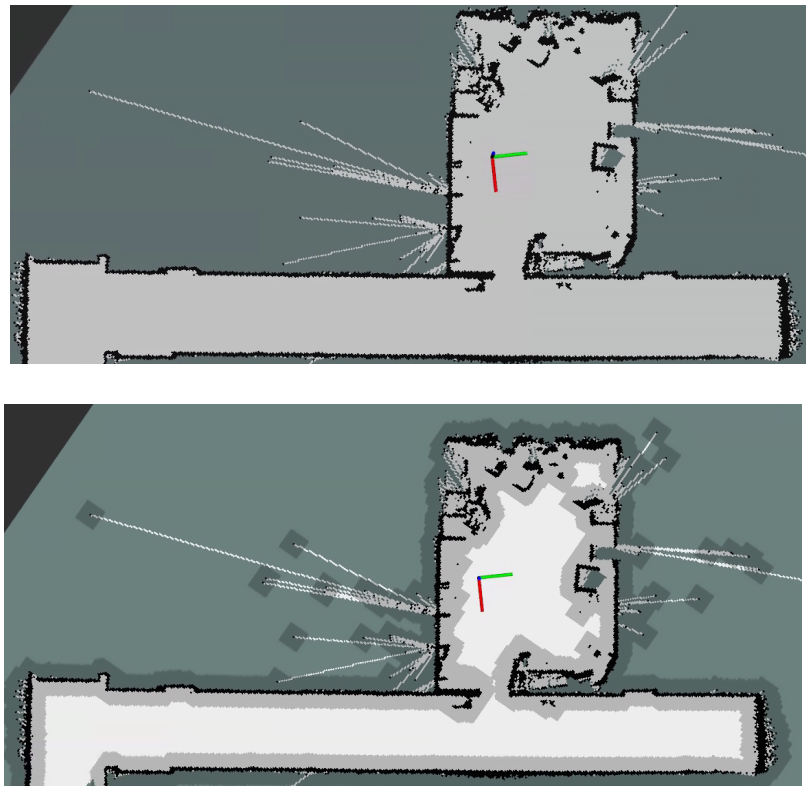


Figure 4.3: Partial Map of the Rutgers University Engineering A-Wing Building (Floor 2) as a result of the Hector SLAM and Obstacle Inflation module

interaction on the map. The user clicked location is subscribed by the path planner as global cartesian co-ordinates.

4.3.1 Dijkstra's Algorithm

The results obstained by Dijkstra's Algorithm are shown in Fig 4.6. Since the path planner is dynamic, it replans the path if there are newly found obstacles on the way. Due to this , the input to the path planner can be from a partially built map or a point from unexplored area of the map. This way a much more robust path planner is obtained. If the robot finds that there are obstacles at the goal co-ordinate then it searches for free space in the vicinity of the goal co-ordinates and replans again. If that is not possible, the path planner terminates. Fig 4.7 shows the autonomous replanning of the path. The controller loop sends the signal to the planner for replanning which

is run at a frequency of 300Hz.

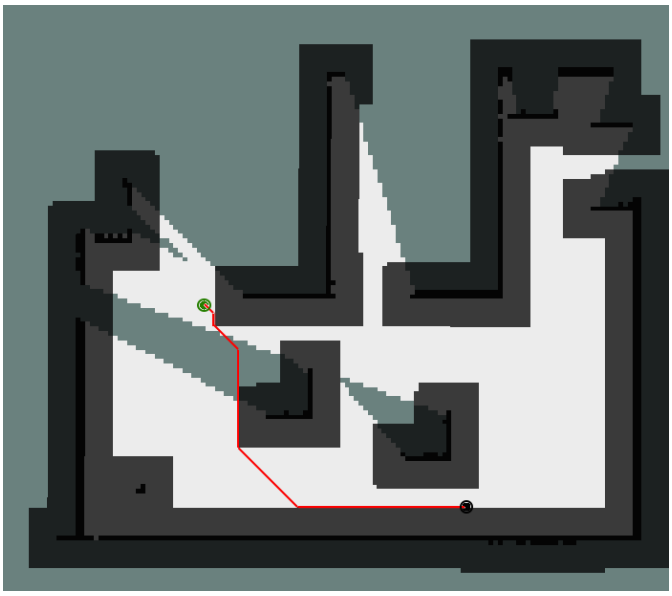


Figure 4.6: Result of Dijkstra's path planner (Black Circle is the start position & Green circle is the goal position)

4.3.2 A* Algorithm

The A* algorithm was implemented with two heuritic distances, Euclidean and Manhattan Heuristic. The results of both in shown in Figure 4.8.

4.3.3 Run Time

The two plots below shows the runtime of the Dijkstra's & A* algorithm w.r.t. to the length of the path. It can be seen that the max run time is close to 16ms for Dijsktra and 4ms for A* algorithm, which makes the path planner well suited for the real time application. As per the aymptotic analysis, Dijkstra's algorithm scales as per the notation $O(VlogV + E)$ with the use of priority queues [34].

Figure 4.7: The replanning process, the top right figure shows that the planner detects an obstacle on the path and replans the path as seen in the bottom left figure. The bottom right figure shows the result of search of new goal position in the vicinity of the original goal for replanning

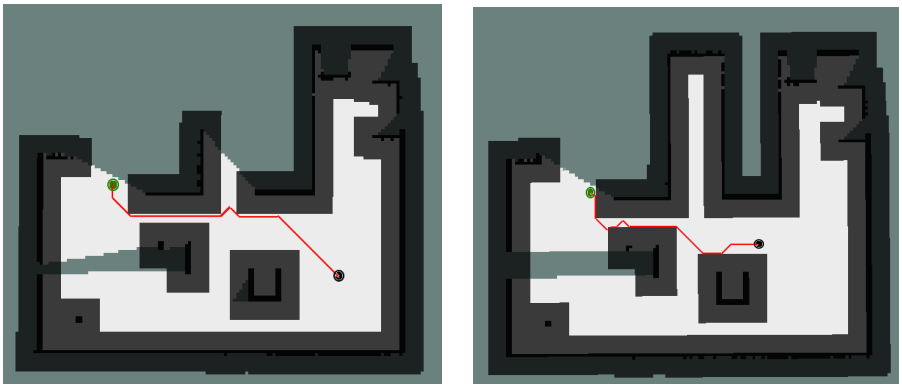
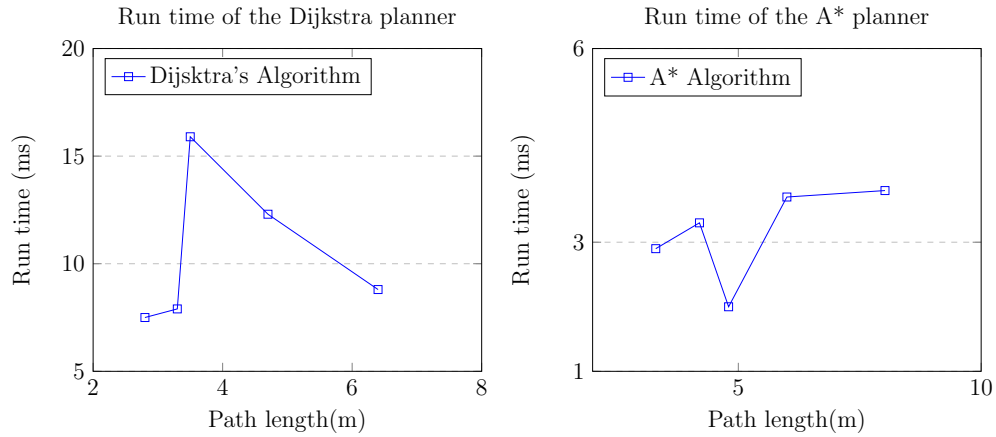


Figure 4.8: A* path planning , left figure shows the result with Euclidean distance as heuristic and the right figure is for Manhattan Distance Heuristic



4.4 Autonomous Exploration

4.4.1 Simulation Results

Figures 4.9 shows the result of the Autonomous Exploration Algorithm. The algorithm stops when path planning is no longer possible. The figures starting the top left figure show the chronological order of the path created by the autonomous exploration module

4.4.2 Experimental Results

Figure 4.10 shows the chronological sequence of the path created by the autonomous exploration algorithm in the lab environment. The last figure is the complete map of the room created at the end of the algorithm.

4.4.3 Run Time

The runtime for the autonomous exploration for finding frontiers , which includes applying sobel operator on the occupancy grid and the finding for the frontiers in the list created by the module , is on average 50ms. The highest computation time observed during the experiments was about 145ms.

4.5 Controller

The experimental results in the lab environment for the control algorithm (Algorithm 6) is shown in Figure 4.11. Due to similarities with differential drive robot, any path can be very closed followed with very little error by seperating linear and angular velocity command. However that is time consuming and hence the combining linear and angular velocity commands is more preferred. A threshold to angular error is introduced. When the angular error is above a threhold , only angular velocity commands are published instead of combined linear and angular velocity command. The maximum controller error observed during the experiments with angle thresholding of $\frac{\pi}{6}$ was 0.05m. This error was added in the obstacle inflation module to avoid any collision with the obstacle due to controller error.

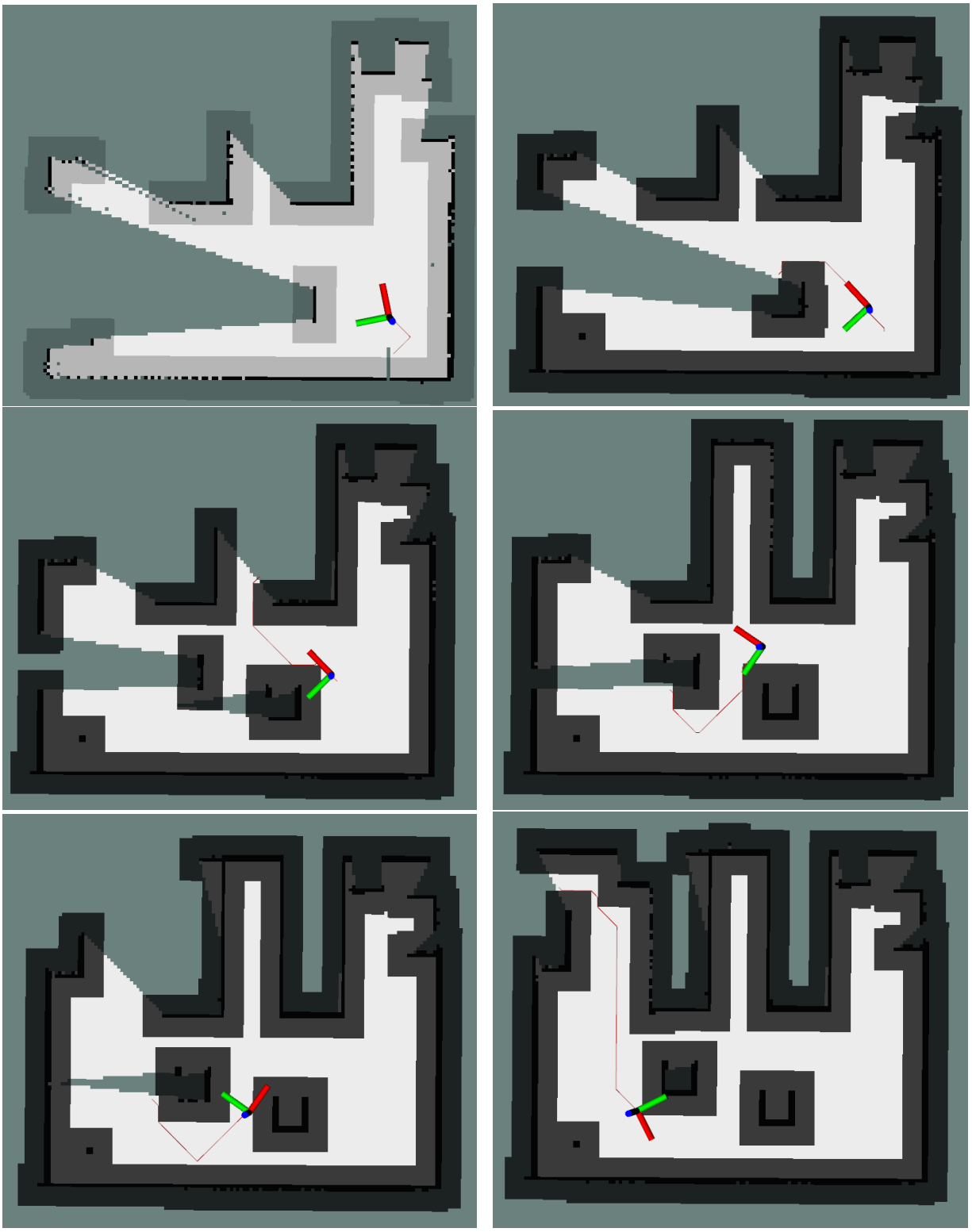


Figure 4.9: Path planning to the Frontiers Detected by the Autonomous Exploration Algorithm

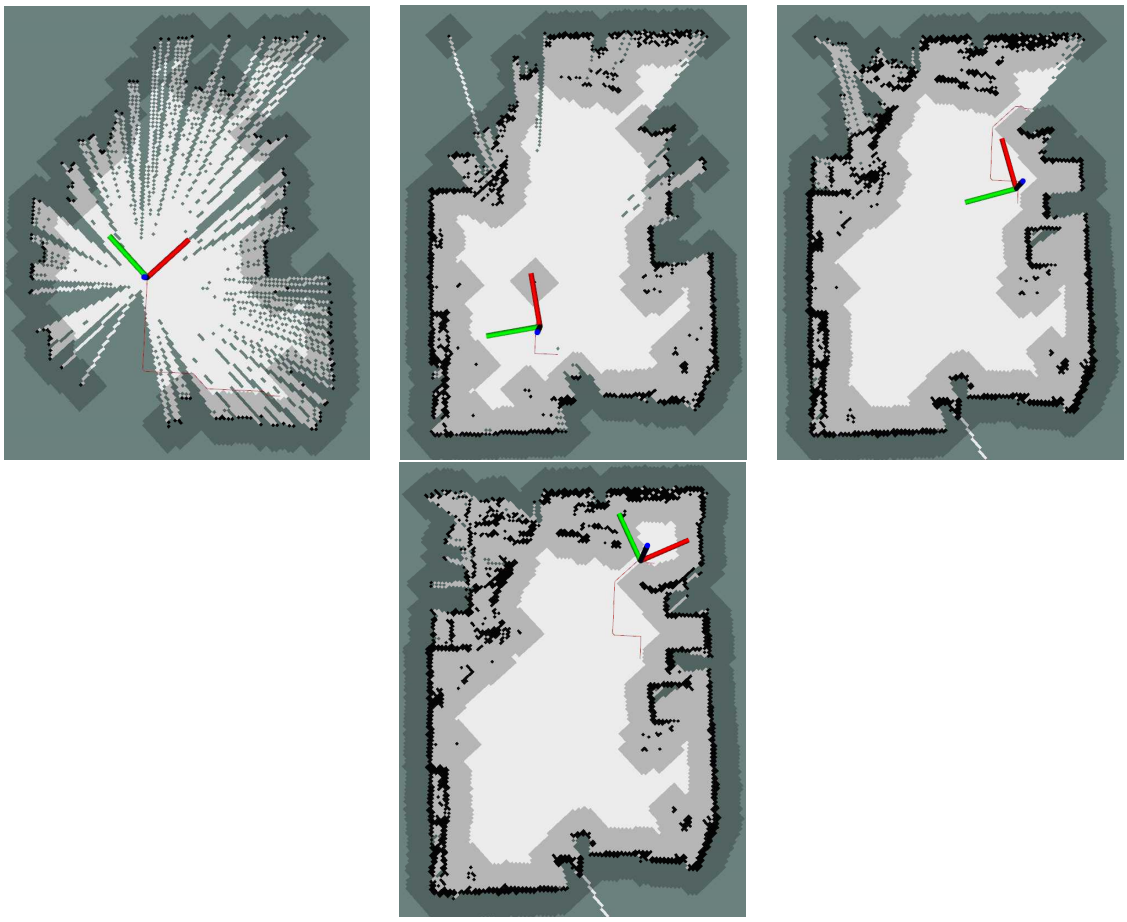


Figure 4.10: Path planning to the Frontiers Detected by the Autonomous Exploration Algorithm in the Lab surrounding

Figure 4.11: Controller Performance , black for is the initial position, red curve is the planned path & blue curve is the actual path

Chapter 5

Conclusion & Future Work

Occupancy Mapping and SLAM

As seen in the results, for relatively smaller areas, Occupancy grid mapping approach is one of the most reliable techniques. However occupancy grid maps can be represented in sparser domains and hence recontructed using Compressed Sensing methods [36]. The idea of compressed sensing is to create the map with smaller number of measurements compared to SLAM modules.

$$y = Hx \tag{5.1}$$

where $y \epsilon R^m$, $x \epsilon R^n$ and n > m ,

y = measurment vector(/signal)

x = vector(/signal) to be contructed from measurments

H = reconstruction matrix

The compressed sensing methods can potentially offer two advantages, compressed representation of large maps for lower memory consumption and map creation using lesser measurments compared to the current SLAM methods.

Obstacle Avoidance

As seen in the section 4.1.1, the obstacle avoidance module is able to integrate the obstacle well in to the occupancy grid and hence allows the path planner to dynamically avoid the obstacle. This increases the field of view of the robot at reduced cost when compared to a 3D LiDAR and is more robust compared to a Sonar. The area of improvement for this module is the computation time. On NVIDIA Jetson TX2 system , the run time of the obstacle detection and integration is on average 1 sec with two cameras. This can be improved further to avoid objects moving faster and closer to the robot. With multiple cameras the use of GPU acceleration is justified for Voxel grid creation. Hence a computationally faster Voxelized grid is being explored as an improvement to obstacle avoidance.

Localization

The robot pose which is the output of SLAM gives good results when there are certain geometric features available in the perceptive range of the LiDAR. For eg. if the LiDAR perceptive range has objects like boxes, water filters etc, the robot pose is generated with lower covariances and the motion accrues fewer errors over time. Experiments carried out in the lab by traversing the robot along a known square as shown in Figure 5.2 showed that the robot error stayed within 0.01m for position and 0.1^o for orientation. However, the error increases considerably in hallways. Figure 5.1 shows such situation in which the the LiDAR's perceptive range is just parallel walls and hence the successive range scans had to localize by comparing only straight lines. The current application of the robot is in relatively smaller rooms, but future application may demand the robot autonomously travel between various rooms and consequently the hallways. This requires the robot to have a robust localization module. One potential solution is to use high frequency LiDAR (40Hz) [1] comapred to

[1] https://www.robotshop.com/en/hokuyo-utm-30lx-ew-laser-rangefinder.html

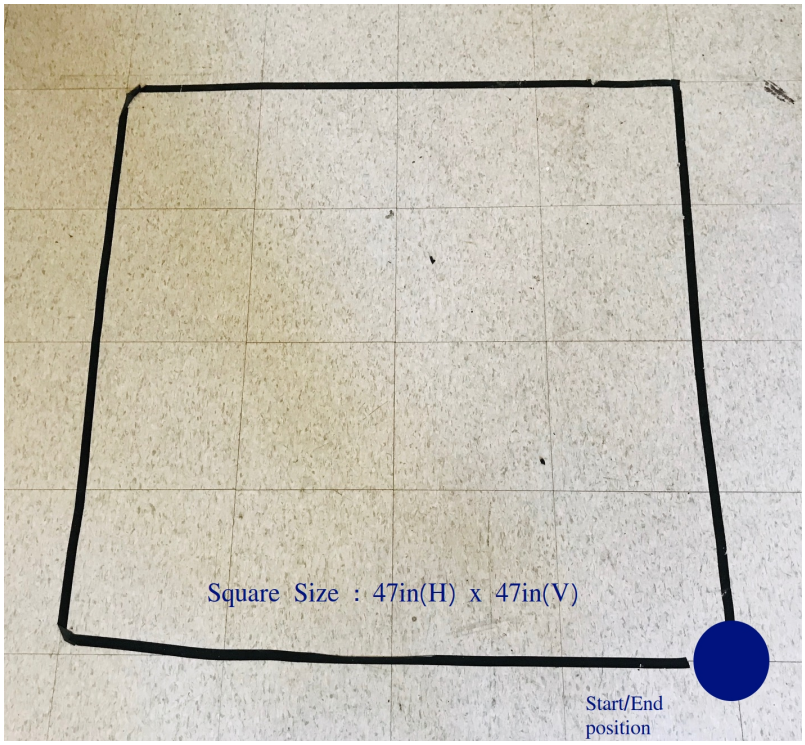


Figure 5.2: Reference Square for testing

current 6Hz sensor. However that is an expensive solution. Another alternative is to use Extended Kalman Filter to fuse the encoder estimated pose with SLAM pose. Due to its accurate kinematic modelling, the torch robot needs to be converted to a differential drive robot to implement EKF based localization. Preliminary experiments were carried out which gave promising result. Mathematical Derivation of EKF for differential drive robot is given in Appendix A.

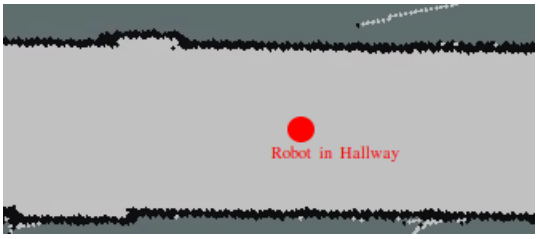


Figure 5.1: Robot in Hallway

Path Planning

The graph based path planning satisfies the realtime path planning requirement for dynamic obstacle avoidance. But a good improvement to this path planner would be post processing for controller error reduction and robust obstacle avoidance. The brief idea for post processing is :

- Sample points from the current path at fixed equally spaced intervals(d)
- Starting from the first point, perform a controller simultation and calculate the point at distance d from the current location
- Perform obstacle check
- If it is obstacle free, replace the sampled point from the planner output with the simulated output
- Repeat the process until the last point in the trajectory

Such post processing is expected to reduce the controller error.

Controller

Another interesting application for GPU acceleration is in using Model Predictive Control (MPC). The optimization for calculating the controller output using MPC can be calculated using GPU. The requirements for this is accurate modelling of the robot dynamics. This has been identified as a potential improvement for the future.

Appendix A

Mathematical Derivations

A.1 Hector Scan Matching

A.1.1 Map Access

For accessing the occupancy value and calculating the gardients, Hector SLAM uses a bilinear filtering based interpolation scheme for higher accuracy. Let P_m be a map co-ordinate. The occupancy value is given as $M(P_m)$ as well as the gradient $\nabla M(P_m) = ((\frac{\partial M}{\partial x})(P_m), \frac{(\partial M)}{\partial y}(P_m))$ can be approximated by using linear interpolation with closest interger co-ordinates $P_{00}, P_{01}, P_{01}, P_{11}$ (figure A.1) as :

$$M(P_m) \approx \frac{y - y_0}{y_1 - y_0}(\frac{x - x_0}{x_1 - x_0}M(P_{11}) + \frac{x_1 - x}{x_1 - x_0}M(P_{01})) + \frac{y_1 - y_0}{y_1 - y_0}(\frac{x - x_0}{x_1 - x_0}M(P_{10}) + \frac{x_1 - x}{x_1 - x_0}M(P_{00})) \quad \text{(A.1)}$$

The gradient of the above equation is given as :

$$\frac{\partial M(P_m)}{\partial x} \approx \frac{y - y_0}{y_1 - y_0}(M(P_{11}) + M(P_{01})) + \frac{y_1 - y}{y_1 - y_0}(M(P_{11}) + M(P_{01}) \quad \text{(A.2)}$$

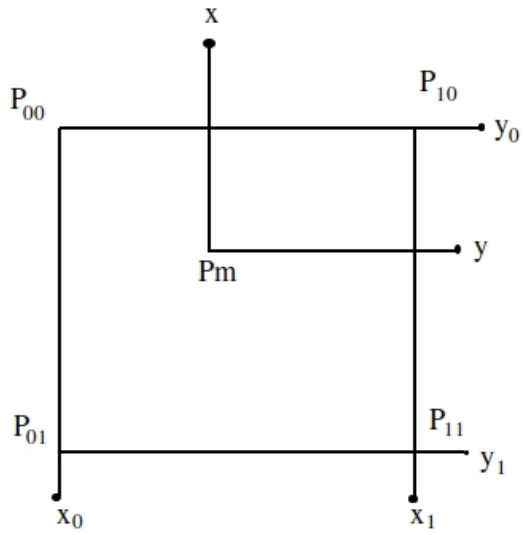


Figure A.1: Closest Integers from the point P_m

$$\frac{\partial M(P_m)}{\partial y} \approx \frac{x - x_0}{x_1 - x_0}(M(P_{11}) + M(P_{01})) + \frac{x_1 - x}{x_1 - x_0}(M(P_{11}) + M(P_{01}) \tag{A.3}$$

A.1.2 Scan Matching

Scan matching is based on the optimization of the alignment of beam endpoint with map obtained so far. Let the current pose of the robot be given by $\xi = (p_x, p_y, \psi)^T$. $S_i(\xi)$ is the location of the point obtained from the laser range scanner. It is given as :

$$S_i(\xi) = \begin{bmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi & cos(\psi) \end{bmatrix} \begin{bmatrix} s_{i,x} \\ s_{i,y} \end{bmatrix} + \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{A.4}$$

$M(S_i(\xi))$ is the occupancy value for the laser beam endpoint. Scan mathcing aims to minimize the error of this occupancy value. Since $S_i(\psi)$ is the distance the beam travelled after reflected from the obstacle. This means that the location of this point on the map must have the occupancy value 1.

$$\xi^* = \underset{\xi}{argmin} \sum_{i=1}^{n} [1 - M(S_i(\xi))]^2 \tag{A.5}$$

Let $\Delta\xi$ be the transformation between the scans which optimizes the equation A.5 according to :

$$\sum_{i=1}^{n}[1 - M(S_i(\xi + \Delta\xi))]^2 \rightarrow 0 \tag{A.6}$$

The taylor expansion of the above equation can be written as :

$$\sum_{i=1}^{n}[2 - M(S_i(\xi)) - \nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}\Delta\xi]^2 \rightarrow 0 \tag{A.7}$$

Setting the partial derivative of the above equation to zero and solving for $\Delta\xi$, we get:

$$\xi = H^{-1}\sum_{i=1}^{n}[\nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}]^T[1 - M(S_i(\xi))] \tag{A.8}$$

where,

$$H = [\nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}]^T[\nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}] \tag{A.9}$$

A.2 Extended Kalman Filter

A.2.1 Odometry Estimate

The current pose of the robot is :

$$\xi = \begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}^T \tag{A.10}$$

Let Δs_r and Δs_l be the distance travelled by the left and the right wheel as per the odometer output. The updated robot pose will be as per equation A.11 .

$$\xi_n = \xi + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} cos(\theta + \frac{\Delta s_r - \Delta s_l}{2b}) \\ \frac{\Delta s_r + \Delta s_l}{2} sin(\theta + \frac{\Delta s_r - \Delta s_l}{2b}) \\ \frac{\Delta s_r - \Delta s_l}{2b} \end{bmatrix} \tag{A.11}$$

Here, b is the distance between the right and the left wheels.

Next step is to establish the co-variance matrix which represents the uncertainty in the pose estimated by the odometers. The encoder error increase in proportional to the distance travelled by the robot. If the wheels rotate faster , then the encoder output will have higher uncertainty. [35] considers various odometry error models and gives a guideline to model systematic and non-systematic errors related to odometry. Considering the uncertainty to be proportional to the distance travelled by the wheels the error matrix will be :

$$\epsilon_q = \begin{bmatrix} k_r|\Delta s_r| & 0 \\ 0 & k_l|\Delta s_l| \end{bmatrix} \tag{A.12}$$

The state space equation for the robot are :

$$x^{'}_{k+1} = f(x_k, u_k) = \begin{bmatrix} x_k + \Delta s cos(\theta + \Delta\frac{\theta}{2}) \\ y_k + \Delta s sin(\theta + \Delta\frac{\theta}{2}) \\ \Delta\theta \end{bmatrix} \tag{A.13}$$

Here , $\Delta\theta = \frac{\Delta s_r - \Delta s_l}{2b}$ and $\Delta s = \frac{\Delta s_l + \Delta s_r}{2}$

The Jacobian of the state-transition function can be evaluated as :

$$F_k = \frac{\partial f}{\partial x_{(x_{k-1}, u_{k-1})}} = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial \theta} \end{bmatrix} \tag{A.14}$$

The **co-variance matrix (Q)** will be :

$$Q = F\epsilon_q F^T \tag{A.15}$$

The **predicted covariance estimate** matrix will be :

$$P_k = F_k P_{k-1} F_k^T + Q \tag{A.16}$$

The initial estimate P_0 can be an identity matrix.

A.2.2 Measurement Update

The measurment update for the EKF will be the robot pose estimated by Scan matching. The measurement model is :

$$z_k = h(x_k) = \begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}^T \tag{A.17}$$

The observation matrix (H) will be the jacobian of the measurment model :

$$H = \begin{bmatrix} \frac{\partial h}{\partial x_k} & \frac{\partial h}{\partial y_k} & \frac{\partial h}{\partial \theta_k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.18}$$

The update equations for the Extended Kalman filter are :

Residual Covariance

$$S_k = H_k P_k H_k^T + R_k \tag{A.19}$$

Here, R_k is the co-variance matrix for the pose measured from scan matching. As per [2] , it can be estimated as :

$$R_k = \sigma^2 H_t^{-1} \tag{A.20}$$

H_t is evaluated as per equation A.9. σ is a scaling factor.

Kalman Gain

$$K_k = P_{k-1} H_k^T S_k^{-1} \tag{A.21}$$

Updated State Estimate

$$x_k = x_k^{'} + K_k(z_k - h(x_k^{'})) \tag{A.22}$$

Updated Covariance Estimate

$$P_k = (I - K_k H_k) P_k \tag{A.23}$$

Bibliography

[1] Brian Yamamuchi, *A Frontier-Based Approach for Autonomous Exploration*, Navy Center for Applied Research in Artificial Intelligence , Computational Intelligence in Robotics and Automation, 1997. CIRA'97

[2] S. Kohlbrecher and J. Meyer and O. von Stryk and U. Klingauf, Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR), *A Flexible and Scalable SLAM System with Full 3D Motion Estimation*, November 2011

[3] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard, *Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling*, In Proc. of the IEEE International Conference on Robotics and Automation (ICRA), 2005

[4] Sebastian Thrun, Wolfram Burgard, Dieter Fox, *Probabilisitic Robotics*, The MIT Press, Cambridge Massachussets

[5] Steven LaValle, *Planning Algorithms*, Cambridge University Press, 2006

[6] Kohlbrecher S., Meyer J., Graber T., Petersen K., Klingauf U., von Stryk O. (2014) *Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots*. In: Behnke S., Veloso M., Visser A., Xiong R. (eds) RoboCup 2013: Robot World Cup XVII. RoboCup 2013. Lecture Notes in Computer Science, vol 8371. Springer, Berlin, Heidelberg

[7] I. Sobel, *An isotropic 33 gradient operator*, in Machine Vision for Three-Dimensional Scenes, H. Freeman, Ed., pp. 376–379, Academic Press, New York, NY, USA, 1990.

[8] Roland Seigward, Illah Nourbaksh, *Introduction to Autonomous Mobile Robots* The MIT Press, Cambridge, Massachussets, 2004

[9] Sebastian Thrun,Robot Mapping:A Survey, CMU-CS-02-111, February 2002

[10] A. Elfes. *Sonar-based real-world mapping and navigation.*IEEE Journal of Robotics and Automation, RA-3(3):249–265,June 1987.

[11] A. Elfes.*Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1989.

[12] H. P. Moravec. *Sensor fusion in certainty grids for mobile robots.AI Magazine*, 9(2):61–74, 1988

[13] M. J. Mataric. *A distributed model for mobile robot environment-learning and navigation*. Master's thesis, MIT, Cambridge, MA, January 1990. also available as MIT AI Lab Tech Report AITR-1228.

[14] B. Kuipers and Y.-T. Byun. *A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations*. Journal of Robotics and Autonomous Systems , 8:47–63, 1991.

[15] F. Lu and E. Milios. *Globally consistent range scan alignment for environment mapping.*Autonomous Robots, 4:333–349,1997

[16] Michael Montemerlo, Sebastian Thrun, Daphne Koller Ben Wegbreit, *FastSLAM:A Factored Solution to the Simultaneous Localization and Mapping Problem*

[17] J.A. Castellanos and J.D. Tard os.*M*obile Robot Localization and Map Building: A Multisensor Fusion Approach. Kluwer Academic Publishers, Boston, MA, 2000.

[18] Lumelsky, V., Skewis, T., *Incorporating Range Sensing in the Robot Navigation Function.* IEEE Transactions on Systems, Man, and Cybernetics , 20:1990, pp. 1058–1068..

[19] Khatib, O., 1985, *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots.* 1985 IEEE International Conference on Robotics and Automation , March 25-28, St. Louis, pp: 500-505.

[20] Koren, Y., Borenstein, J., *High-Speed Obstacl Avoidance for Mobile Robotics,* in Proceedings of the IEEE Symposium on Intelligent Control,Arlington, VA, August 1988, pp: 382-384.

[21] Borenstein, J., Koren, Y., *The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots.* IEEE Journal of Robotics and Automation , 7, pp: 278–288, 1991.

[22] Ulrich, I., Borenstein, J., *VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots,* in Proceedings of the International Conference on Robotics and Automation (ICRA'98) , Leuven, Belgium, May 1998

[23] Ulrich, I., Borenstein, J., *VFH*: Local Obstacle Avoidance with Look-Ahead Verification,* in Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, May 24–28, 2000

[24] Khatib, O., Quinlan, S., *Elastic Bands: Connecting, Path Planning and Control,* in Proceedings of IEEE International Conference on Robotics and Automation , Atlanta, GA, May 1993

[25] Dijkstra, E. W. (1959). *A note on two problems in connexion with graphs*. Numerische Mathematik. 1: 269–271.

[26] Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics SSC4. 4 (2): 100–107.

[27] Don Murray, Cullen Jennings(1996), *Stereo vision based mapping and navigation for mobile robots*, Department of Computer Science, University of British Columbia, Vancouver, ICRA.

[28] Lavelle, Kuffner, *Rapidly-Exploring Random Trees: A New Tool for Path Planning*

[29] N.M. Amato , Y. Wu, *A randomized roadmap method for path & manipulation planning*, IEEE International Conf., Robotics & Automation, pg. 113-120 (1996)

[30] Jonathan D. Gammell, Siddhartha S. Srinivasa, Timothy D. Barfoot, *Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic*, 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014), pp. 2997-3004

[31] . J. Kuffner and S. M. LaValle. *An efficient approach to path planning using balanced bidirectional RRT search*. Technical Report CMU-RI-TR-05-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2005

[32] Thomas Edlinger, Ewald Von Puttkamer, *Exploration of an Indoor-Environment by an Autonomous Mobile Robot*, IROS 1994

[33] Cyril Stachniss, Wolfram Burgard, *Exploring Unknown Environments with Mobile Robots using Coverage Maps*

[34] Fredman, Michael Lawrence; Tarjan, Robert E. (1984). *Fibonacci heaps and their uses in improved network optimization algorithms*. 25th Annual Symposium on Foundations of Computer Science. IEEE. pp. 338ndash, 346. doi:10.1109/SFCS.1984.715934.

[35] Borenstein, J., Everett, H.R., Feng, L., *Where Am I? Sensors and Methods for Mobile Robot Positioning*. Ann Arbor, University of Michigan

[36] *Real Time SLAM Using Compressed Occupancy Grids For a Low Cost Autonomous Underwater Vehicle*, Christopher Cain, PhD Dissertation, Virginia Institute of Technology ,2014

[37] Pearl, J. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, 1984. p. 48.