Obstacle Detection and Mapping using Lidar for Self-driving car applications

Archana Ramalingam
Computer Engineering Department, San Jose State University, San Jose CA USA 95192
archana.ramalingam@sjsu.edu

***Abstract* - In today's busy world, people are required to complete more tasks in less time. Technology has now paved a path in such a way that, machines are taking control of major time consuming things. Autonomous vehicles technology is one such application of the future. While autonomous cars find use among common people, unmanned vehicles can save lives in military. The capability to sense the surrounding, learn from the pattern and control the car without human input is the basic essence. The use of sensors is indispensable in autonomous cars. Sensor system for self-driving cars use sophisticated sensors like LIDARs, RADARs, cameras, night vision (infrared cameras), stereo vision, ultrasonic sensors, etc. The surrounding area of the self-driving car is mapped using Lidar. The map is generated as grids based on probabilistic reasoning. Based on this grid map, the obstacles in the path of the car are determined. In this paper, we are implementing occupancy grid mapping using a RPLIDAR, Hector mapping on a ROS (Robot Operating System). We are also looking to solve the challenges faced by the Hector mapping algorithm.**

***Key Words* – Self-driving cars, Perception sensors, LIDAR, ROS, Occupancy grid map, Path Planning algorithms, Hector SLAM**

I. INTRODUCTION

A Self-driving car, like any other computer system has three basic blocks – Input, Processing and Output. The Input for the Self-driving cars is typically the Perception/Sensing System which includes Sensors like Lidar, Radar, Cameras, Ultrasonics, etc. The sensors indicate the presence of obstacles in the environment around the car. The Processing part includes advanced algorithms for Mapping and Path Planning (Trajectory). Finally, the Output typically includes the Motor/Actuation System which drives the car in the longitudinal (acceleration & braking) and lateral (steering) directions in the given trajectory. In this project, the Sensing System of the self-driving car is studied and implemented. The Lidar sensor is used to scan the environment. A 2D Map is created from the Lidar scans and represented in the Grid Map format suitable for processing in the computer.

II. SYSTEM OVERVIEW

The System consists of Hardware Sensor, Software Framework for development and the Processing Algorithms for Grid Map representation.

A hardware sensor (LIDAR) is used for the perception of the environment around the sensor. The environment might typically contain obstacles which are detected by the sensor and avoided while the car/robot is driving around.

The location of the obstacles is efficiently represented in Occupancy grid map format for further processing by the path planning algorithms.

The entire software for this project will be developed based on the open-source ROS (Robot Operating System) framework.

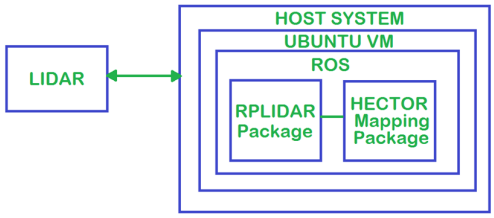

Fig.1. Project layout

III. THE SENSOR

The LIDAR (Light detection and ranging) sensor used for this project is low cost RPLIDAR 360° 2D Laser Scanner. It performs 360 degree laser scanning up to 6 meters distance detection range. The produced 2D point cloud data is used in Mapping and Environment modeling in Grid Map format.

As shown in the figure 2, the Lidar outputs the distance value (d) to the object in front for each

corresponding scan angle value (θ) complete for all 360 degrees, at the rate of 5.5 Hz, generating about 2000 sample points per second.

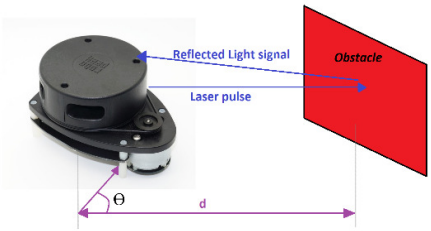

Fig.2. LIDAR operation

IV. SOFTWARE FRAMEWORK

ROS (Robot Operating System) is an open source software to control robotic components from a PC. It is used for both research and commercial purposes. It acts as a framework to write robotic applications by providing the necessary tools and libraries to software developers. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management and more. A dedicated RPLidar package is available in ROS for integrating and working with this particular Lidar inside the ROS framework. Hector SLAM ROS package implements the hector mapping algorithm to generate an occupancy grid map.

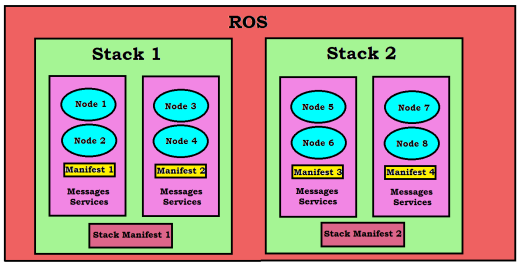

Fig.3. ROS structure

V. OCCUPANCY GRID MAP

An occupancy grid map is used to represent the location of these obstacles on a two-dimensional coordinate plane.

The whole environment around the Lidar is represented as 2D grids and they are marked as occupied spatially or not.

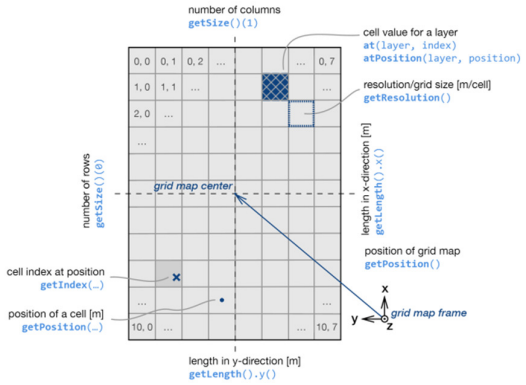

Fig.4. Occupancy Grid map representation

VI. ALGORITHM RESEARCH

When using laser scan data to plot maps, important factors to remember are that: robot's position should be estimated, a high dimensional environment should be mapped, map should be updated as the real world changes. Most algorithms use probabilities to increase robustness to noise and formal representation of uncertainty. Graph based SLAM techniques explored in this project are: Hector SLAM, Gmapping, Costmap_2d ROS packages.

Gmapping uses Rao-Blackwellized particle filer in order to construct the grid maps. Each particle has a map environment, hence more particles increase the accuracy of the map. It requires odometry data. The reason this will not suit us is that, we are doing hand-held mapping with Lidar, which means we do not have wheel encoders or IMU to generate odometry data for this algorithm, which will otherwise not work without odometry.

Costmap fills the grid map with discrete values in the range of 0 to 255, with 255-no information, 254-occupied, 0-not occupied. As it uses discrete value for each cell, the map resolution is poor. There is no special computation done to calculate the cell value, which makes it simple, but not practical for real-time applications as Autonomous Vehicles.

VII. HECTOR SLAM

This algorithm focuses on robot movement estimation in real-time, taking advantage of the high update rate and low distance measurement noise from modern LIDARs. Unlike Gmapping, the odometric information is not required, which

makes its usability wider into aerial robots like, a Quadrotor UAV, ground robots in uneven terrains and also in hand-held Lidar, used in this project. Also unlike costmap_2d, the grid values are not discrete, as Hector SLAM uses interpolation to calculate sub grid probabilities too (shown in Map update below) and uses Gauss-Newton approach to match, map and update successive laser scans (shown in Scan Matching below). This makes Hector SLAM independent of odometry data and provide better map resolution. Hence we have chosen Hector SLAM algorithm to be used in this project.

We will look into how the algorithm works. It performs 3 main functions. **Scan transformation** converts the scan data to map frame. The 2d point cloud data from Lidar is converted to array of grids. To improve the precision, we need to have continuous values and this is done by **map update** by linearly interpolating values of the neighboring grid values to find the sub-grid cell value. Hence the grid has more values than a regular occupancy grid, hence making it more accurate.

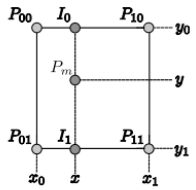

Fig.5. Point P_m occupancy is calculated by interpolation

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

Scan matching is done to align successive laser scans to the original scan. This makes use of high scan rates and low noise of modern lasers. It optimizes the beam end point alignment with earlier map. We need to find the rigid transformation $\xi = (px, py, \psi)^T$ by minimizing equation 2.

$$\boldsymbol{\xi}^* = \underset{\boldsymbol{\xi}}{\operatorname{argmin}} \sum_{i=1}^{n} [1 - M(\mathbf{S}_i(\boldsymbol{\xi}))]^2 \quad (2)$$

Applying Taylor's expression yields Gauss-Newton equation to minimization problem.

$$\Delta\boldsymbol{\xi} = \mathbf{H}^{-1} \sum_{i=1}^{n} \left[\nabla M(\mathbf{S}_i(\boldsymbol{\xi})) \frac{\partial \mathbf{S}_i(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right]^T [1 - M(\mathbf{S}_i(\boldsymbol{\xi}))] \quad (3)$$

VIII. IMPLEMENTATION

We are using Ubuntu Xenial and ROS Kinetic Kame. After installing ROS in Ubuntu and cloning the RPLidar and Hector_slam ROS packages from Github, we then launch Lidar and Hector SLAM together using the following command:

```
roslaunch rplidar_ros view_slam.launch
```

By listening to 'scan' topic, the Lidar data is mapped using Hector SLAM algorithm. The first function of scan transformation is performed by the following function:

```
bool HectorMappingRos::
rosLaserScanToDataContainer (const
sensor_msgs::LaserScan& scan,
hectorslam::DataContainer&
dataContainer, float scaleToMap)
```

Interpolation is performed by:

```
float interpMapValue(const
Eigen::Vector2f& coords)
```

The covariances and Hessian derivatives for scan matching are calculated using:

```
void getCompleteHessianDerivs(const
Eigen::Vector3f& pose, const
DataContainer& dataPoints,
Eigen::Matrix3f& H, Eigen::Vector3f&
dTr)
```

IX. RESULTS

However, when the Lidar makes a sharp turn, the mapping loses the position of the Lidar and starts mapping from a new position. The primary reason is getting stuck in local minimas. In order to solve this problem, we have handled multiple solutions. The prompt also prints the error: "SearchDir angle change too large". We will modify the parameters in order to overcome this problem. The parameters are put together in the corresponding launch file. We will decide on the optimized set of parameters depending on the minimum number of times the error is logged and also maximum amount of time it takes for the first error to be logged. Also only results that displayed noticeable improvement are included in this report.

1. Setting maximum **searchdir** to higher than the default value 0.2, eg: 0.2*8 - increases Searchdir angle value, so that a sudden turn does not throw an exception of the angle being too large, as the defined value is large enough already.

2. Increasing the **map_multi_res_levels** - increases the number of maps used.
3. Decreasing the **map_resolution** – reduces effect of noisy readings.
4. Decreasing **map_update_distance_threshold** - reducing the amount of distance the robot can move per iteration.
5. Decreasing **map_update_angle_thresh** - reducing the angle the robot can move per iteration
6. Increasing **number of iterations** – more iterations yields to better mapping.

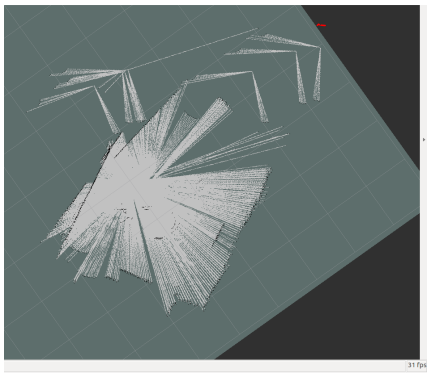

Fig.6. *map_resolution* = 0.01

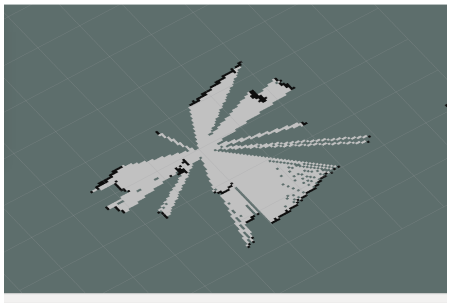

Fig.7. map_multi_res_levels = 7

From the above experiments, a set of parameter values were selected which perform optimized mapping that reduced the error to some extent. The parameters used were:

searchdir = 0.2, map_multi_res_levels = 7,
map_resolution = 0.01,
map_update_distance_threshold = 0.02,
map_update_angle_thresh = 0.1

Other solutions that work are using a pre-recorded ROS bag file to map or using Gmapping, where the problem is negligent as it uses particle filters.

Other challenges faced are that the LIDAR behaves differently with objects of different materials and colors.

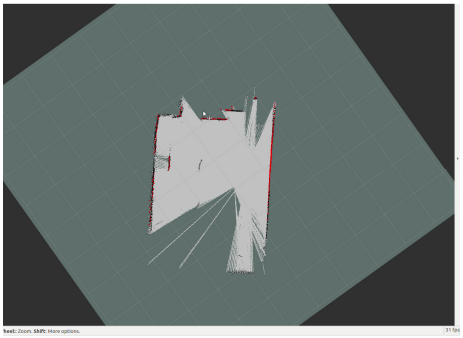

Fig.8. Map with optimized set of parameters

X. CONCLUSION

- The main challenge faced in hector mapping is the sensor position being lost due to the sharp movements of the Lidar. For a hand-held sensor, this does not pose much problems, however if the sensor is mounted on a constantly moving robot, then the problem is bigger.
- LIDAR is a well suited sensor for obstacle detection
- ROS is the preferred middleware to manage sensors
- Hector mapping is best method to map the environment into an Occupancy grid map using LIDAR.

XI. FUTURE WORK

- Once the obstacles are detected, a path is plotted by the self-driving car avoiding the obstacles using advanced path planning algorithms
- Can be scaled to 3D Lidar
- SLAM (simultaneous localization and mapping) – extending the project to perform SLAM by integrating IMU.

XII. REFERENCES

1. http://library.isr.ist.utl.pt/docs/roswiki/Robots(2f)Nao.html
2. http://blog.pal-robotics.com/
3. http://www.active8robots.com/services/programming/
4. http://www.slideshare.net/AtlayMayada/ros-based-programming-and-visualization-of-quadrotor-helicopters
5. http://wiki.ros.org/ROS/Tutorials

6. http://wiki.ros.org/rplidar

7. http://robohub.org/ros-101-intro-to-the-robot-operating-system/

8. http://www.distraction.gov/stats-research-laws/facts-and-statistics.html

9. https://www.intoxalock.com/ignition-interlock-devices/statistics

10. https://prezi.com/ploqna92fvds/hector-slam-system-diagram/

11. Stefan Kohlbrecher, Oskar von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable SLAM system with full 3d motion estimation. In Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on, pages 155–160. IEEE, 2011.

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

13. N. Sariff, N. Buniyamin, An Overview of Autonomous Mobile Robot Path Planning Algorithms, 2006, 4th Student Conference on Research and Development.

14. Martin Buehler, Karl Iagnemma, Sanjiv Singh,Springer, 2009, The DARPA Urban Challenge: Autonomous Vehicles in City Traffic.