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

Team 6

Archana Ramalingam

SJSU ID: 010761114

Guided by

Prof. Kaikai Liu

CMPE Department

San Jose State University

Autonomous Vehicles (AV)

Leading Motivations

- Reduced accident rates
- In USA, in 2014 alone,
    - Distracted driving: 431,000 injured, 3179 killed
    - Drunk driving: 9967 killed
- Increased Productivity – avg. 50 mins/day
- Empower the Specially abled
- Parking – 15% tighter space

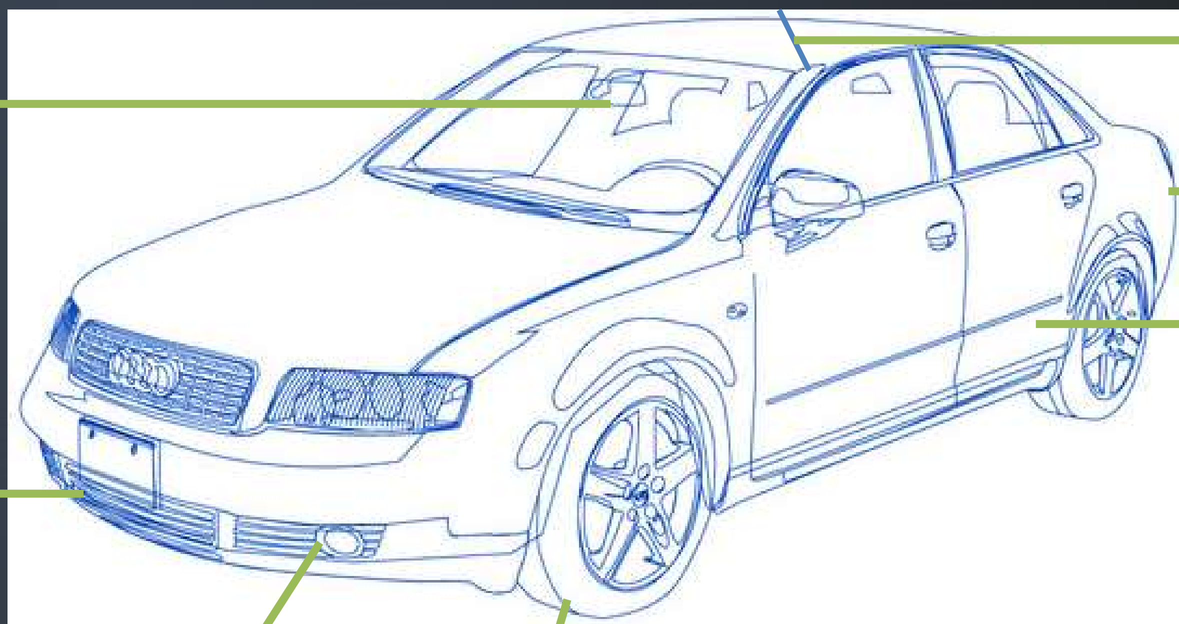
How far have we come?

- In USA, National Highway Traffic Safety Administration classification:
  - Level 0 : 100% Driver controlled
  - Level 1 : Stability control, Automatic braking
  - Level 2 : Adaptive cruise control & Lane keeping
  - Level 3 : Mostly Automated, Driver take-over needed occasionally (2018-2020)
  - Level 4 : 100% (fully) Automated

They are coming!

- In June 2011, Nevada became the first state to legalize Autonomous vehicles. (Assembly Bill no. 511)

How do they work?



Camera

GPS

Ultrasonic Sensors

Odometry Sensors

LIDAR

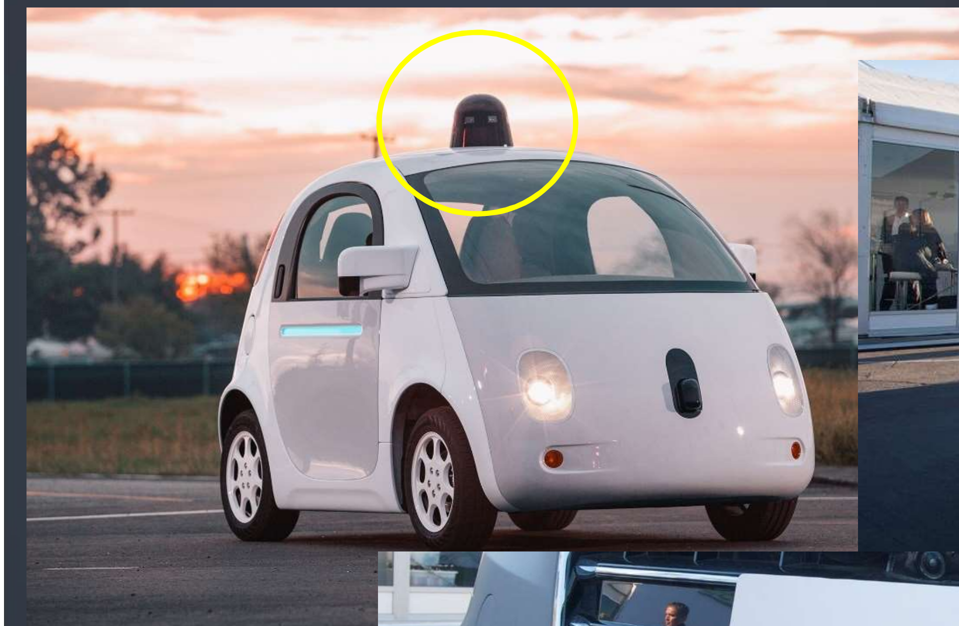
RADAR

Wheel Encoders

What the car sees?

Courtesy: Google

On the road



Project Overview

- A 2D LIDAR (Light Detection And Ranging) sensor is used to scan the environment.
- A 2D Map is created from the Lidar scans and represented in the Occupancy Grid Map format suitable for processing in the computer.
- Used Hector Mapping algorithm to generate map and optimized it.

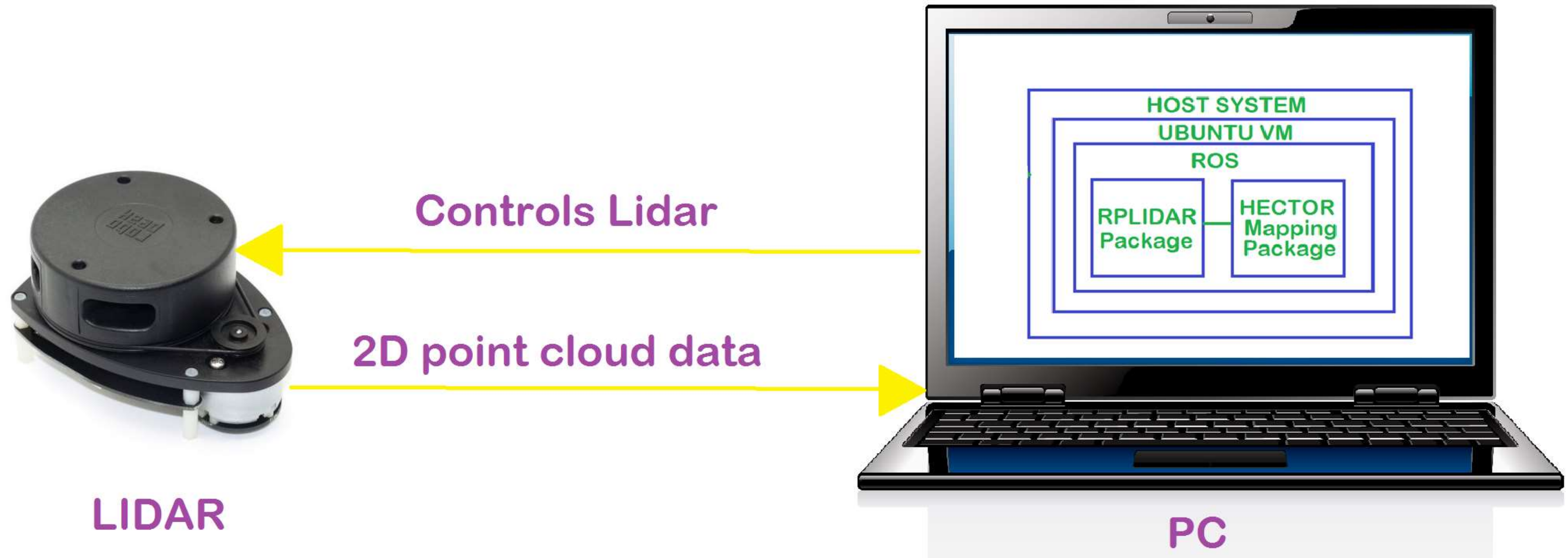
Introduction

- Basic blocks of a Self-driving car
  - Input, Processing and Output
- Input - Perception/Sensing System
  - Sensors like Lidar, Radar, Cameras, Ultrasonic, Night vision (infrared cameras), stereo vision
- Processing - Advanced algorithms for Mapping and Path Planning
- Output - Motor/Actuation System

System Overview

- Hardware Sensor (LIDAR)
- Software Framework for development (ROS)
- Processing Algorithms for Grid Map representation

Design Layout



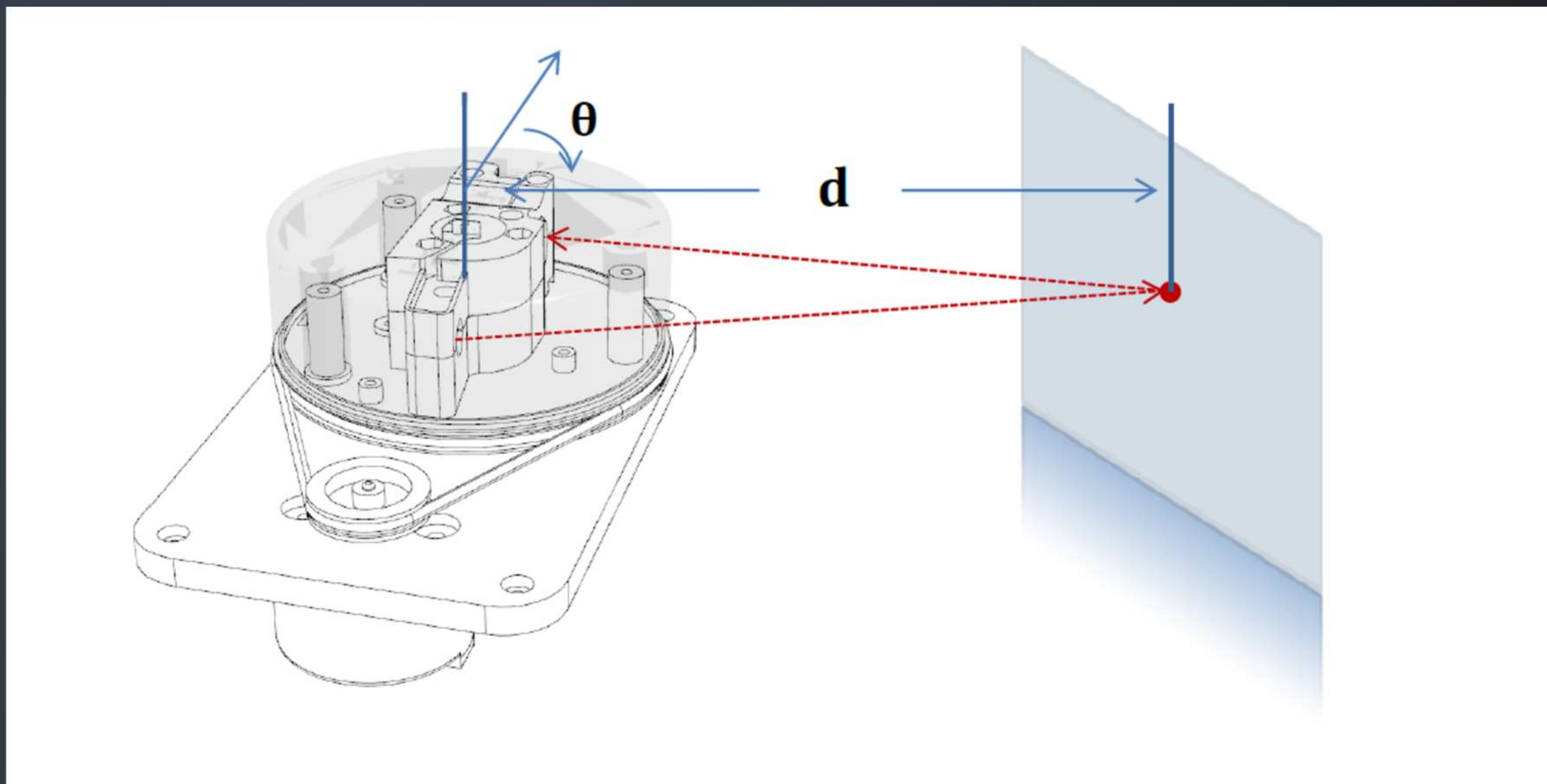
Why LIDAR?

- High scanning frequency
- Class I laser safety standard
- Accuracy & Reliability
- Ignores Sunlight and indoor light
- Better than Ultrasonic, RADAR

LIDAR

- 360 degree 2D laser scanner - RoboPeak
- Laser, Scanner and Photodetector
- Laser triangulation ranging principle
- Distance resolution- < 0.5 mm
- Angular resolution- < 1 degree
- Distance Range-0.2m to 6m
- Sample Frequency- >=2000 samples/second
- Scan rate- 5.5 Hz

LIDAR



Courtesy: RoboPeak

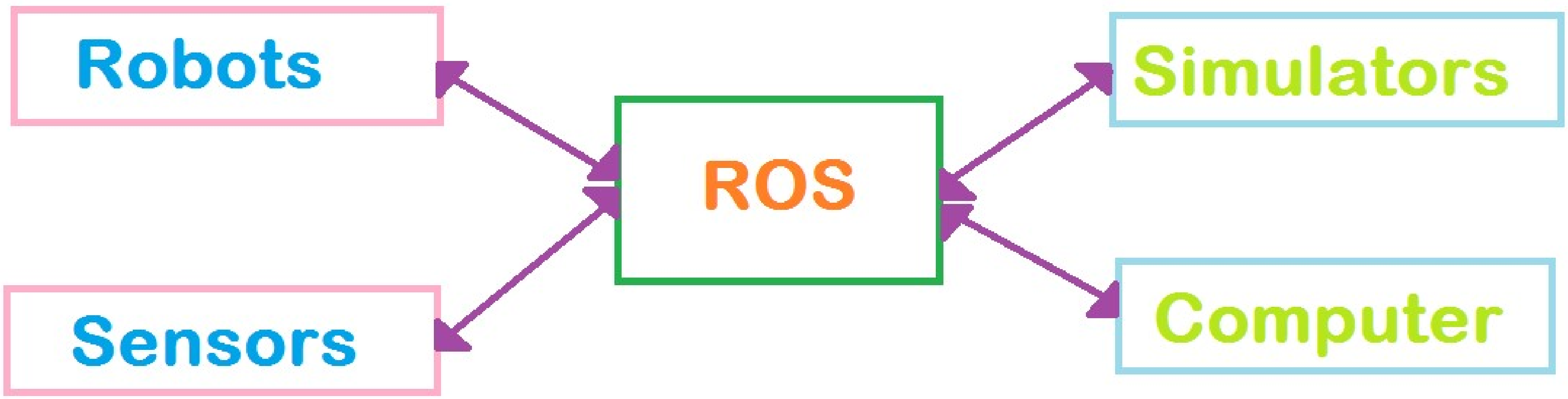
Software Framework

- ROS (Robot Operating System)
- Open-source middleware/meta OS
- Provides hardware abstraction, low-level device control, message-passing between processes, package management

Why ROS?

- Open source software.
- Extended using the numerous packages.
- Overlaying.
- Huge user community
- Supports nodes with different architecture
- Flexible and simple.

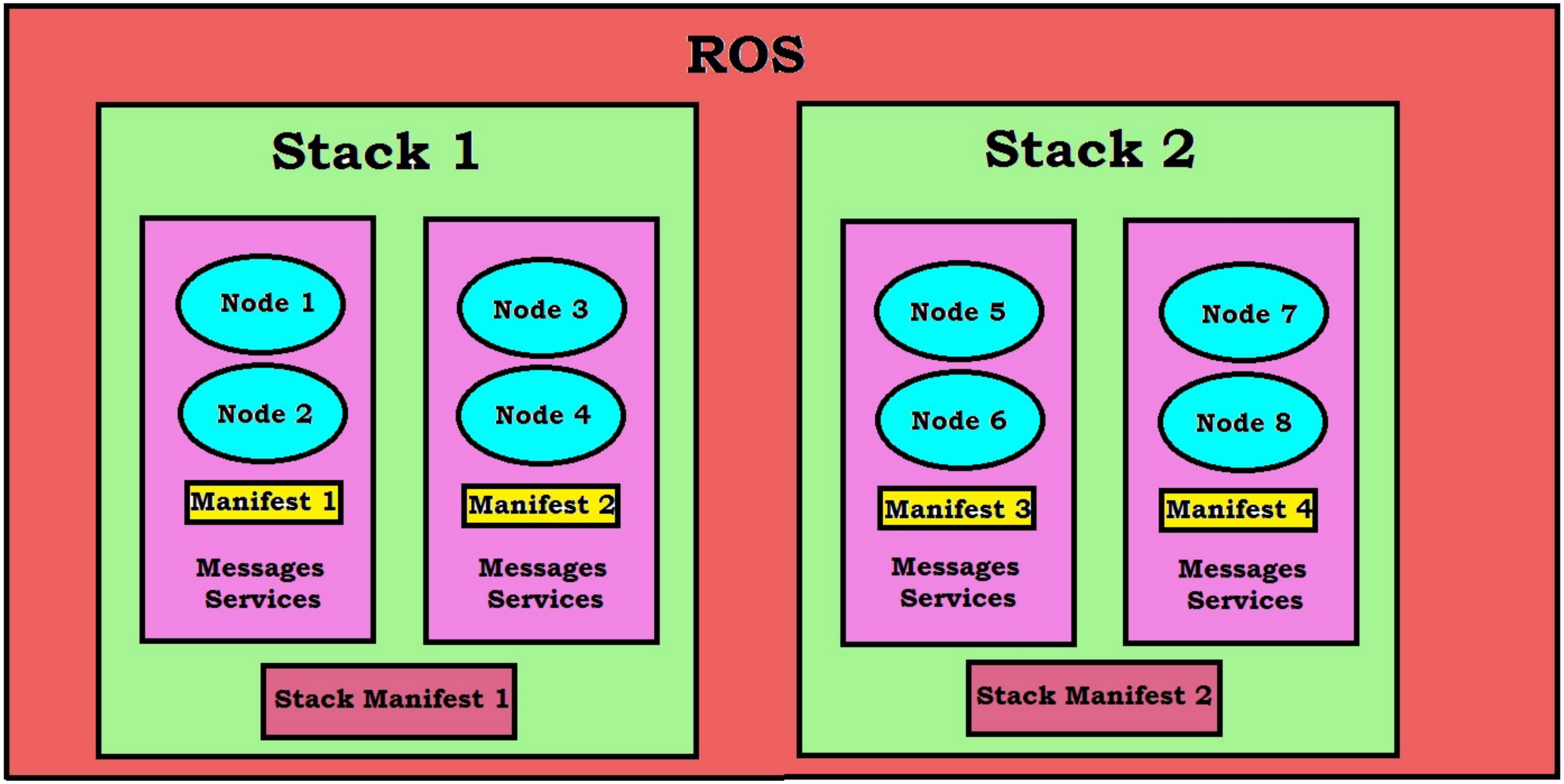
ROS Layout



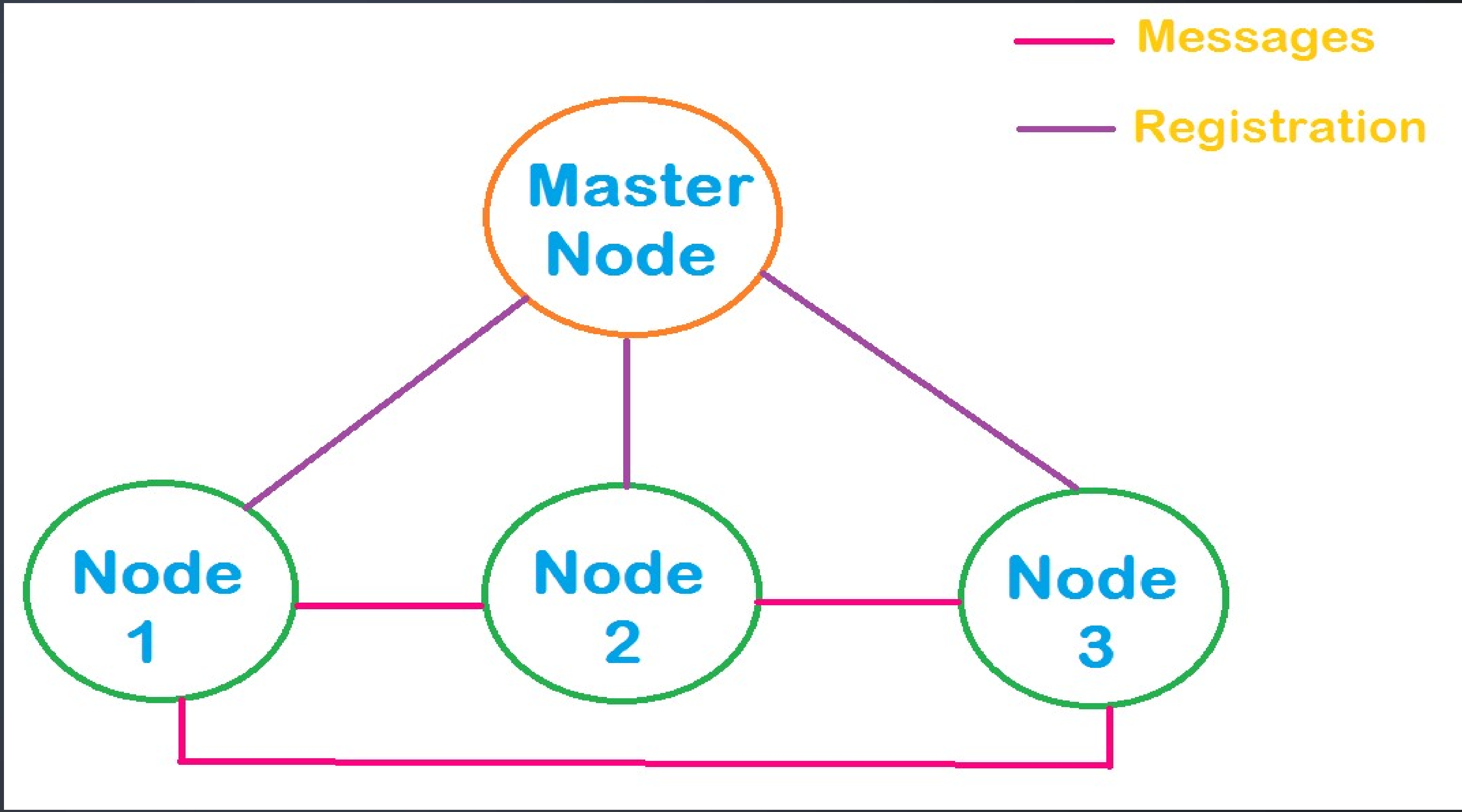
ROS components

- Publisher/subscriber – peer-peer networking
- Package
- Topics
- Messages
- Services
- Manifest – package.xml
- Master Node

ROS Structure



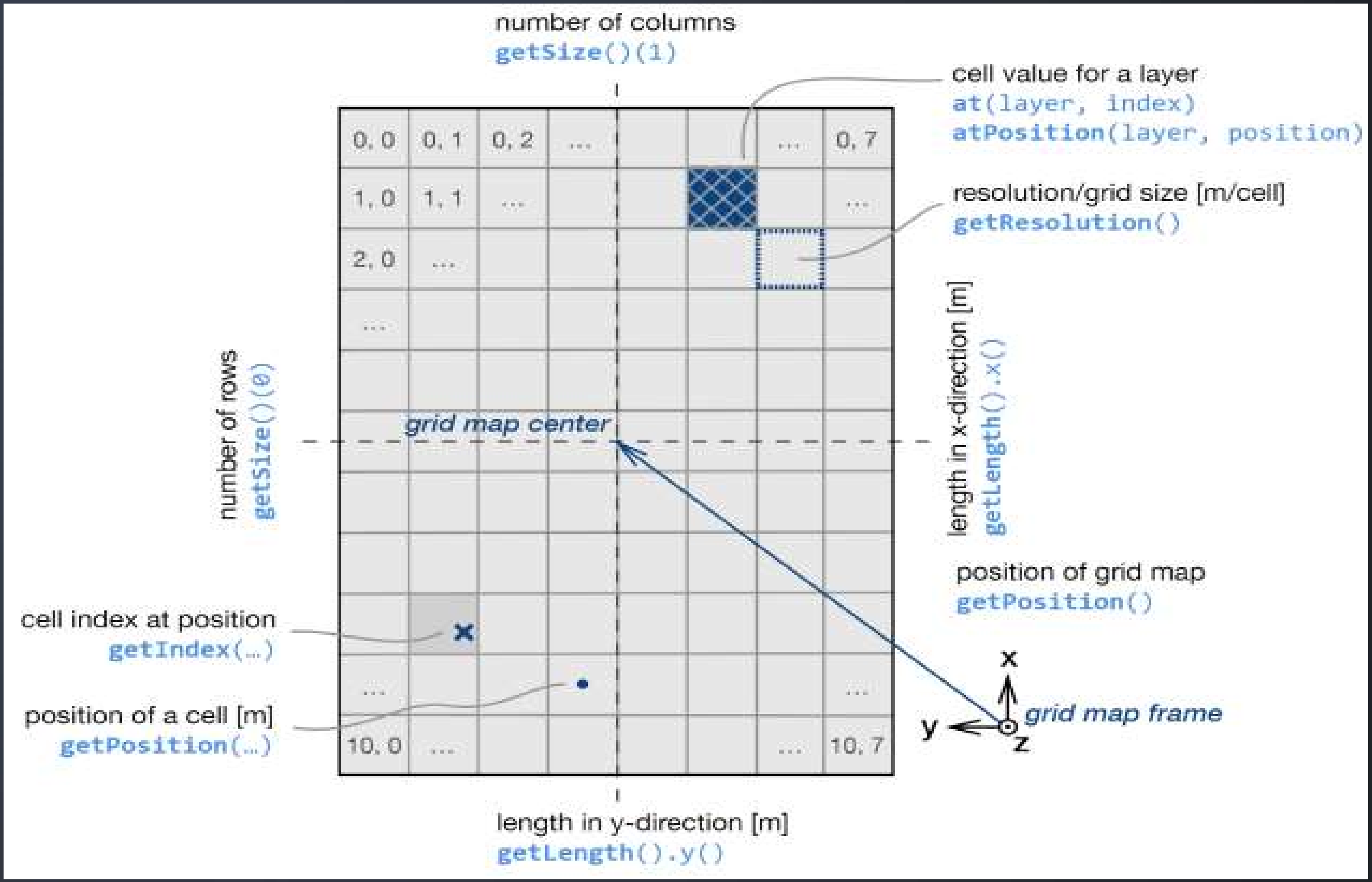
Node Structure



Occupancy Grid Mapping

- An algorithm in Probabilistic Robotics
- Addresses problem of map generation from noisy sensor data
- The map is represented as series of grids with indices
- If g_i is a grid cell of index i, then $p(g_i)$ denotes the probability that the particular cell is occupied

Occupancy Grid Mapping



Algorithm research

- Gmapping
  - Rao-Blackwellized particle filer
  - Cons - Requires odometry data – Wheel encoders, IMU
- costmap2d
  - 0-255 – grid cell values range
  - 255-no information, 254-occupied,
    0-not occupied
  - Cons - discrete value for each cell gives poor map resolution

Algorithm research

- Hector Mapping
  - Odometry data not required
  - Map Update – makes grid values continuous
    - Linear Interpolation

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

  - Scan Matching – makes use of Lidar's high scan rates – aligns successive laser scans
    - Gauss- Newton approach

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

Implementation

- Ubuntu Xenial
- ROS Kinetic Kame
- Rplidar package
- Hector Slam package
- Rviz – ROS visualization

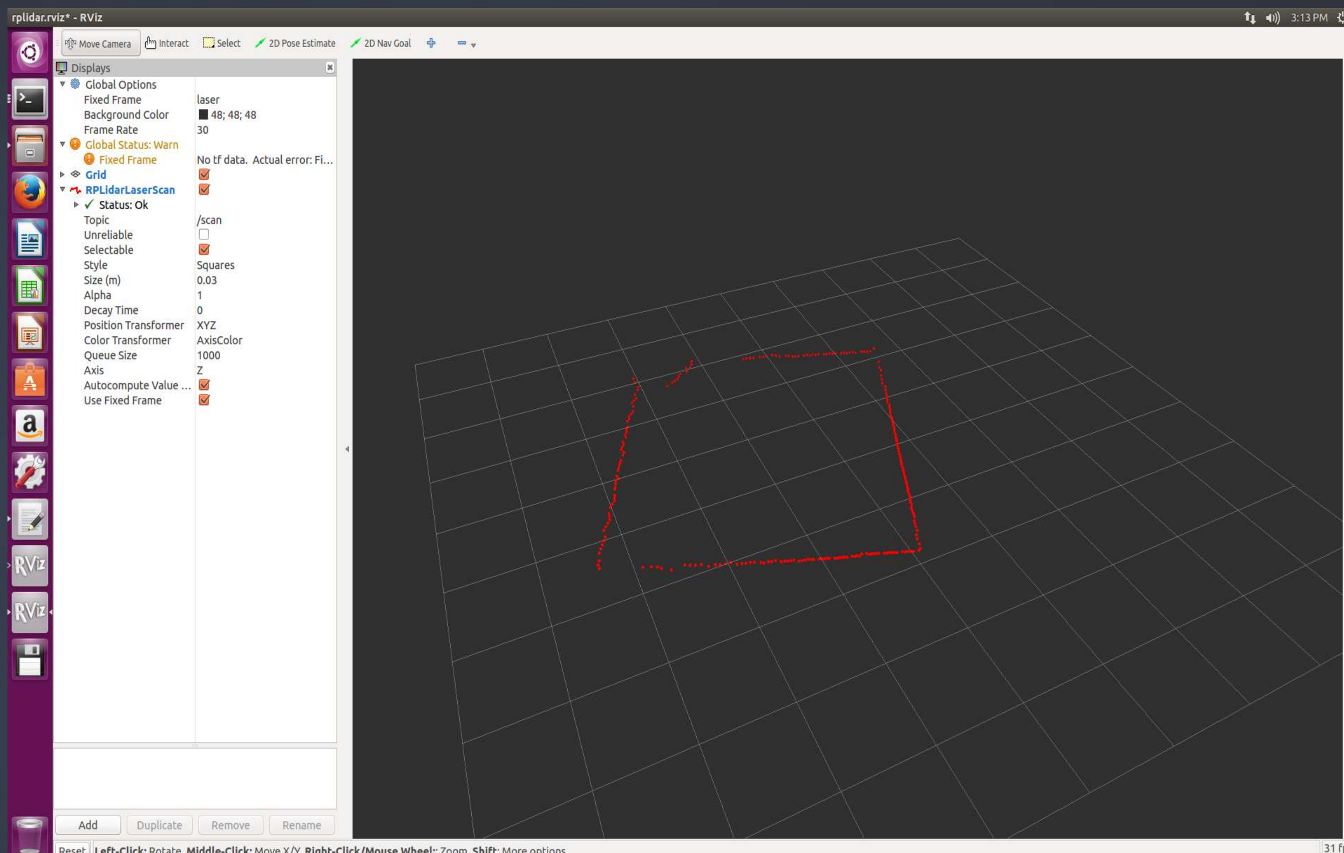
Implementation

- Install ROS in Ubuntu:
  - `sudo apt-get install ros-kinetic-desktop-full`
- Clone RPLidar ROS package
  - `git clone https: //github.com /robopeak/rplidar_ros.git`
- Clone Hector Slam package
  - `git clone https://github.com/tu-darmstadt-ros-pkg/hector_slam.git`
- To launch Lidar and Hector Slam
  - `roslaunch rplidar_ros view_slam.launch`

Implementation

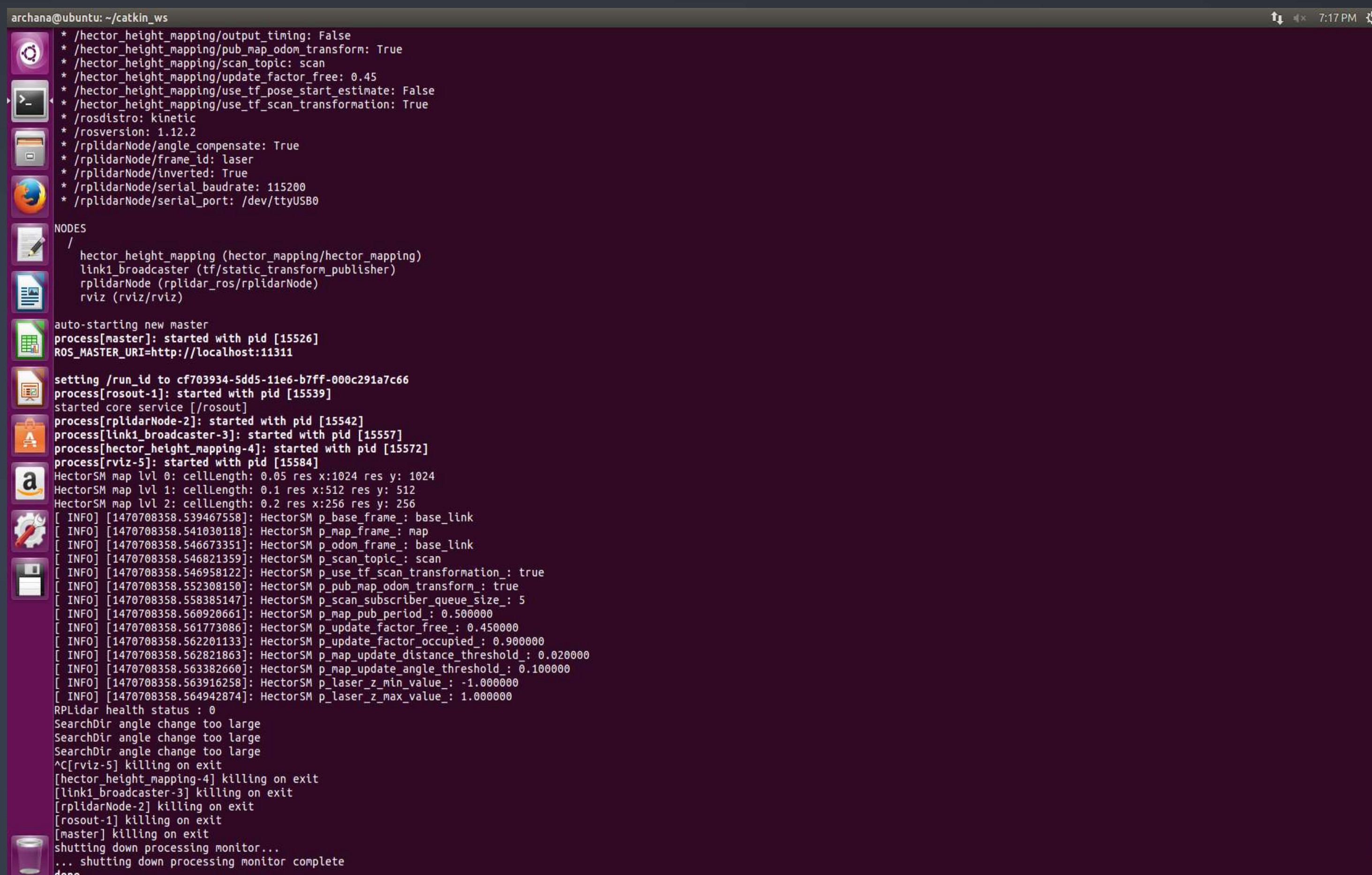
Raw Lidar scan data mapped in Rviz

```
roslaunch rplidar_ros view_rplidar.launch
```



Implementation

`roslaunch` initiating master node and setting parameters for hector mapping



Implementation

// Linear Interpolation for Map update in Hector Slam

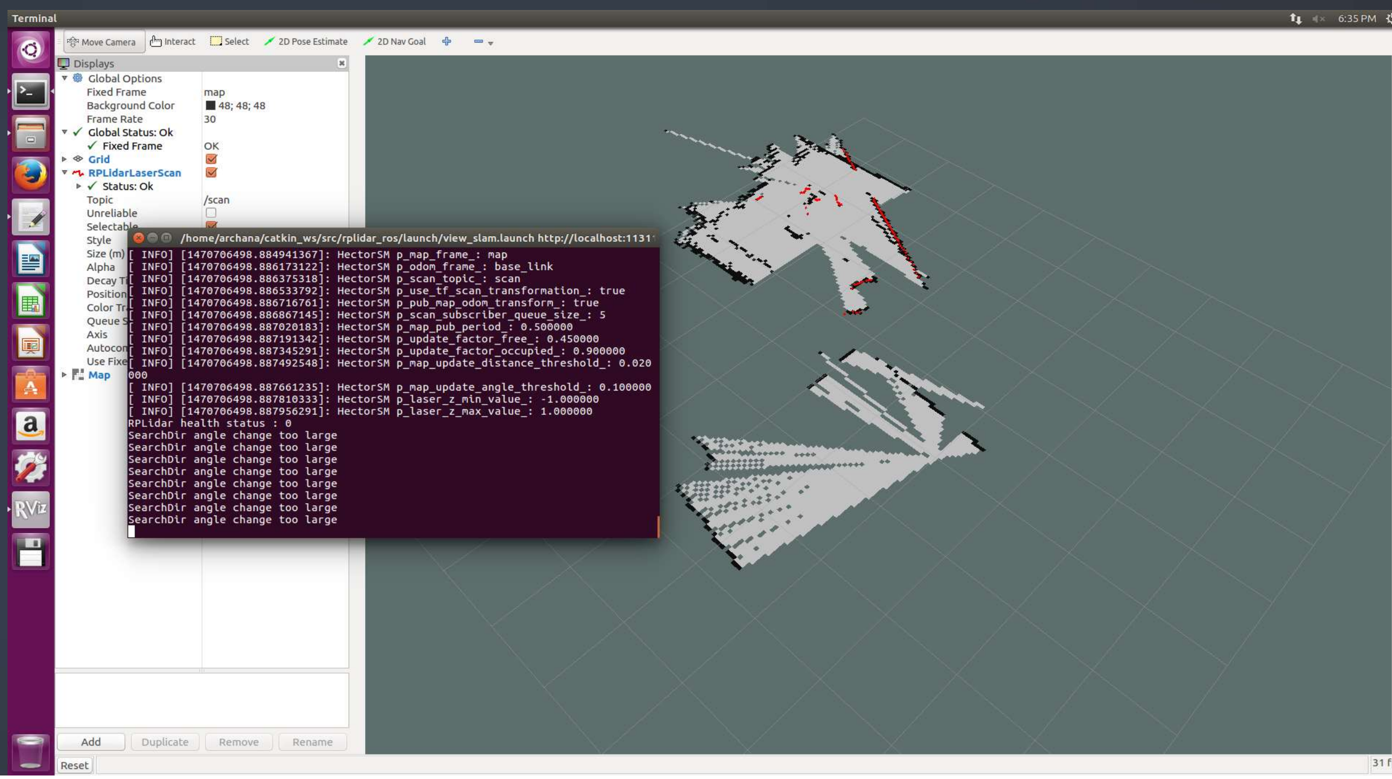
```
float interpMapValue(const Eigen::Vector2f& coords)
  {
    //check if coords are within map limits.
    if (concreteGridMap->pointOutOfMapBounds(coords)){
      return 0.0f;
    }
    //map coords are alway positive, floor them by casting to int
    Eigen::Vector2i indMin(coords.cast<int>());

    //get factors for bilinear interpolation
    Eigen::Vector2f factors(coords - indMin.cast<float>());
    int sizeX = concreteGridMap->getSizeX();
    int index = indMin[1] * sizeX + indMin[0];

    // get grid values for the 4 grid points surrounding the current coords. Check
cached data first, if not contained
    // filter gridPoint with gaussian and store in cache.
    if (!cacheMethod.containsCachedData(index, intensities[0])) {
      intensities[0] = getUnfilteredGridPoint(index);
      cacheMethod.cacheData(index, intensities[0]);
    }
```

Implementation

- For Scan matching, covariances and hessian derivatives are calculated in the following function. (OccGridMapUtil.h)
  - ```
    void getCompleteHessianDerivs(const Eigen::Vector3f& pose, const DataContainer& dataPoints, Eigen::Matrix3f& H, Eigen::Vector3f& dTr)
    ```
- Update the Occupancy grid (OccGridMapBase.h)
  - ```
    /*@param dataContainer Contains the laser scan data
    *@param robotPoseWorld The 2D robot pose in world coordinates*/
    void updateByScan(const DataContainer& dataContainer, const Eigen::Vector3f& robotPoseWorld)
    ```

Implementation

- Challenges faced: Due to Lidar sharp movement
  - 'SearchDir angle change too large' error - getting stuck in local minimas

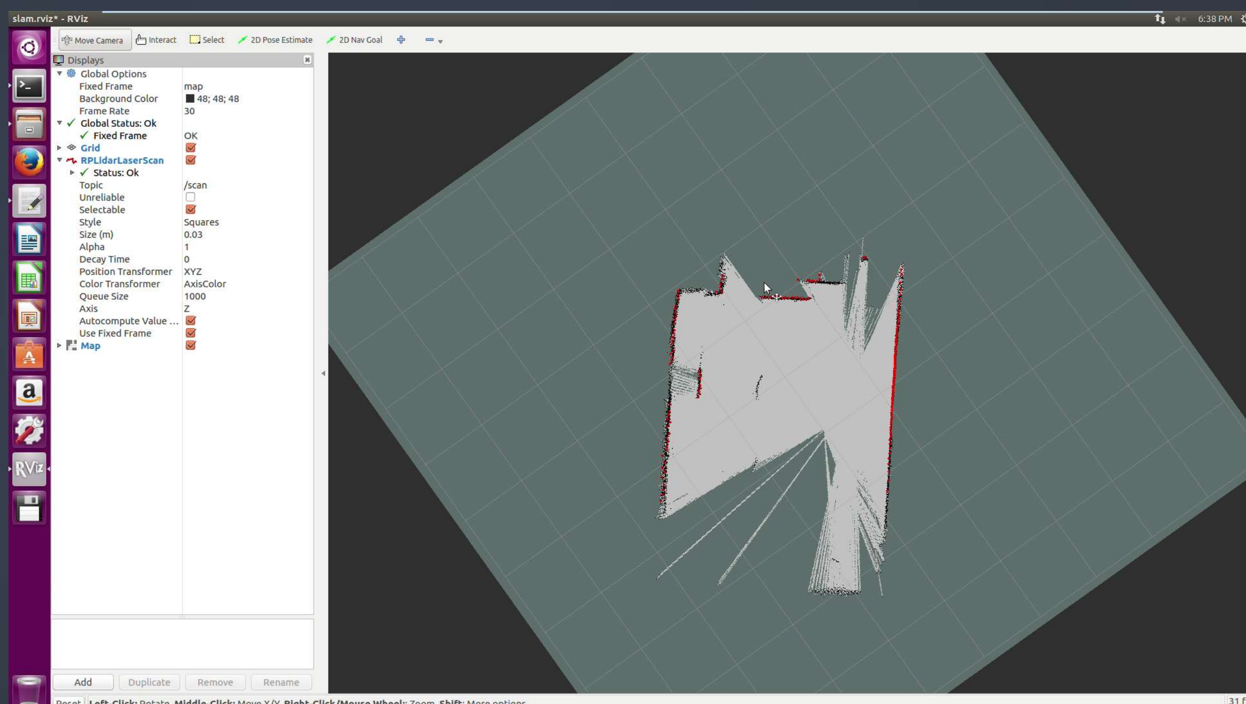


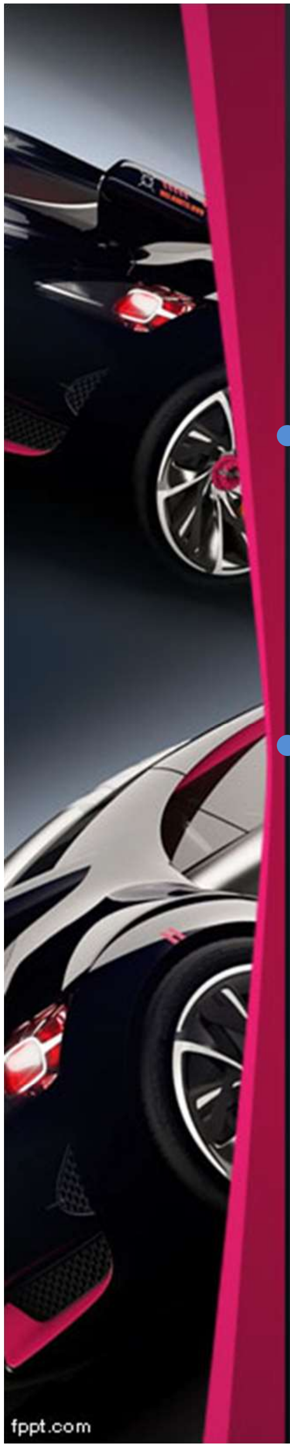
Implementation

- Solutions handled:
  - Setting maximum searchdir to higher than the default value 0.2, eg: 0.2*8
  - Increasing the map_multi_res_levels
  - Decreasing the map_resolution
  - Decreasing map_update_distance_threshold
  - Decreasing map_update_angle_thresh

Results

- Optimized parameters used: searchdir = 0.2, map_multi_res_levels = 7, map_resolution = 0.01, map_update_distance_threshold = 0.02, map_update_angle_thresh = 0.1





Implementation

- Other solutions that work:
  - Using recorded bag file to map
  - Gmapping - problem is negligent (particle filters)
- Other challenges:
  - LIDAR behaves differently - objects of different materials and colors.
  - Exploring algorithms used with hand-held Lidar sensor - depth research.
  - Learning ROS from scratch.

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.

Conclusion

- Main challenge faced in hector mapping - sensor position being lost due to the sharp movements of the Lidar.
- 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.
- Map update rate of Hector Slam - faster than Gmapping – slowed by particle filters

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. https://www.youtube.com/watch?v=MD255BS0YH4
8. http://robohub.org/ros-101-intro-to-the-robot-operating-system/
9. http://www.distraction.gov/stats-research-laws/facts-and-statistics.html
10. https://www.intoxalock.com/ignition-interlock-devices/statistics
11. https://prezi.com/ploqna92fvds/hector-slam-system-diagram/
12. 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.
13. 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.

Thank you!!