6.146 MASLAB LIDAR WHITEPAPER

eboehlke@mit.edu, adim@mit.edu, johnz@mit.edu

December 2019

1 What is LIDAR

LIDAR is a time-of-flight distance sensor. LIDAR is short for "Light Detection and Ranging" and is sometimes referred to as laser scanning. Infrared laser light is shined on a target and the reflected light is measured by a sensor. A point cloud can be constructed by analyzing the laser return time and wavelengths. LIDAR sensors are used in autonomous self-driving cars and robots. Recently, LIDAR sensors have become more affordable increasing their prevalence in hobby robotics. They have become cheap enough that even poor MASLAB students can afford them.

2 The YDLIDAR

The YDLIDAR is one such purveyor of cheap LIDAR sensor. In the MASLAB kit bots, we have included a YDLIDAR X4. The X4 is a planar LIDAR meaning all measurements are made in a single 2D plane. Thus, the measured points are reported with two coordinates e.g. (x, y). The X4 samples at 5000 times per second and has a 11 meter maximum scanning range. Most critically, it costs only $99 on Amazon.

2.1 Wiring up the X4 YDLIDAR

The X4 comes with the LIDAR itself as well as a breakout board allowing a USB micro connection to a computer. This is the recommended way of interfacing with the X4. There are also four screws provided to secure your X4 to your robot. The X4 is powered over USB. The breakout board is connected to the X4 by a 8-pin JST connector.

3 Getting a ROS Message

The RobotShop product listing for the X4 YDLIDAR has lots of information about how to use the LIDAR. `https://www.robotshop.com/en/ydlidar-x4-360-laser-scanner.html`

To use the LIDAR with ROS, download the ROS zip file from the RobotShop website and put in in your catkin workspace. The package says it works with ROS Kinetic but it also works with Melodic which is what we are using for MASLAB 2020. Then make your workspace to build the package.

Figure 1: The YDLIDAR X4. Almost as good as the Turbo Encabulator, but twice as cheap!

```
cd ~/catkin_ws
catkin_make
```

To start the LIDAR and view the scan images in RVIZ run the following command:

```
roslaunch ydlidar lidar_view.launch
```

To just start the LIDAR without opening RVIZ, run the `lidar.launch` file. This launch file starts a node that publishes `LaserScan` messages to the `/scan` topic. It also creates a transformation from `base_footprint` to `laser_frame`.

4 Localizing Robots with LIDAR

Now you understand what a LIDAR is and what kind of information you can get out of one (namely point-cloud data of your environment), it's time to figure out what we actually DO with this data. Robots in the real world tend to use laser-scanners for many things including object-recognition and localization in unknown environments. We will focus on the latter application in this course.

Localization in unknown environments refers to finding the position of a robot in, for example, a room, when the robot doesn't know the dimensions or shape of that room.

WE can clearly see where the robot is, what direction it is pointing, what shape the room is and etc. But remember the robot knows none of these things (we didn't even give it a map). So the question becomes, with the sensors we have how do we get the robot not only to understand what the area it is traversing looks like but where it is in that environment? That is, how do we get the robot to build a map, and then locate itself in the map at the same time?

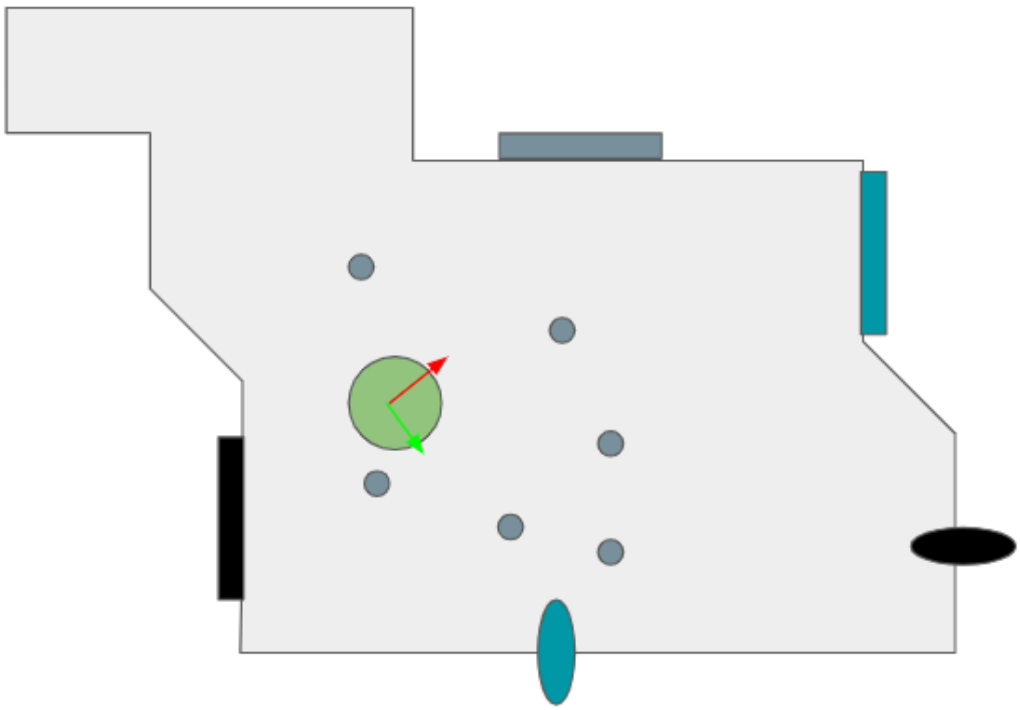

Figure 2: Example room where the green circle is the robot.

4.1 SLAM: Simultaneous Localization and Mapping

SLAM stands for Simultaneous Localization and Mapping and it is an algorithm developed to answer both the questions we asked in the above section. As the robot drives around, SLAM takes in the sensor data and builds a map of the environment it is traversing and then estimates the robots position in that environment as an (x, y, θ) position which includes the location and direction of travel of the robot.

4.2 How does SLAM Work?

SLAM can be done with many different sensors including stereo-depth cameras, 3D LIDARS, 2D LIDARS, with and without odometery/etc. We will focus on 2D LIDAR SLAM with scan-matching odometery to explain the concept.[1]

4.2.1 SLAM Inputs

First, we need to understand the inputs of SLAM before we can understand how SLAM works. SLAM takes, in general, two inputs. First, is some sort of 2D/3D sensor scan or input (in our case the LIDAR) which gives SLAM information about the environment. LIDAR inputs are called pointclouds as described above and are usually inputted to SLAM

[1]A very good overview of SLAM is located here: `https://dspace.mit.edu/bitstream/handle/1721.1/119149/16-412j-spring-2005/contents/projects/1aslam_blas_repo.pdf`

in the form of the full scan. The second, is an input that describes the movement of the robot. We call this odometery and this usually comes in terms of an (x, y, θ) in terms of how the robot has moved based on feedback from things like the robots encoders or similar sensors.

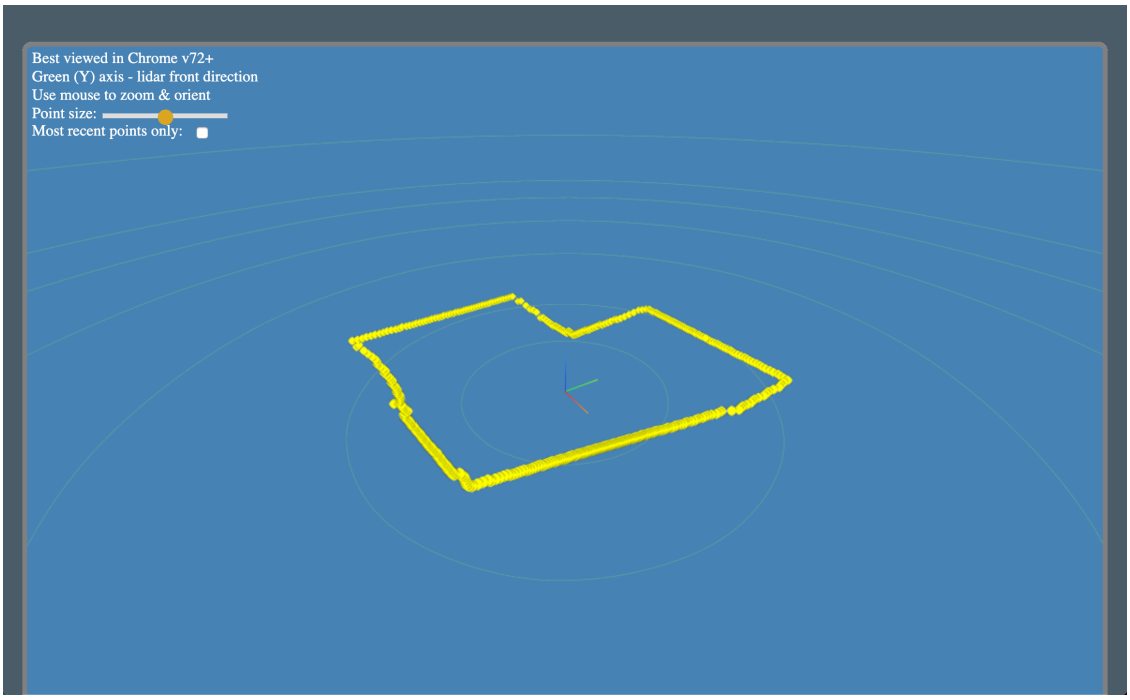

Figure 3: Pointcloud visualization from `http://wayneparrott.com/projects/rplidar-driver/api-docs/`

Figure 3 shows a visualization of a LIDAR point cloud. SLAM will take this point cloud as an input for information on the environment. The point-cloud shows certain features such as walls and tables of the room, walls most distinctly. Figure 4 shows an example of robot odometery. Let's say the robot was at some known position "Previous robot pose." Some control action is executed and the robot is now at "New robot pose." Odometery takes the data from the robots wheels, for example, and says "okay I turned by $\delta rot2$ degrees, I was at $\delta rot1$ degrees and I moved by some $\delta trans$ according to my wheel encoders." Odometery does NOT guess the robot's position, it simply tells slam HOW the robot THINKS it has moved. THINKS because the robots sensors are not perfect and the guessed movement based on the sensors on the wheels are not usually accurate. If they WERE accurate there's no reason for SLAM or any similar algorithm. We'd just use odometery. Remember, the whole reason we need many of these estimation algorithms are because our sensors cannot perfectly capture the real world.

4.2.2 The Algorithm at a High Level

Now that we understand what SLAM takes in, how does it take these things, create a map, and give us a position estimate? The steps in SLAM are as follows:

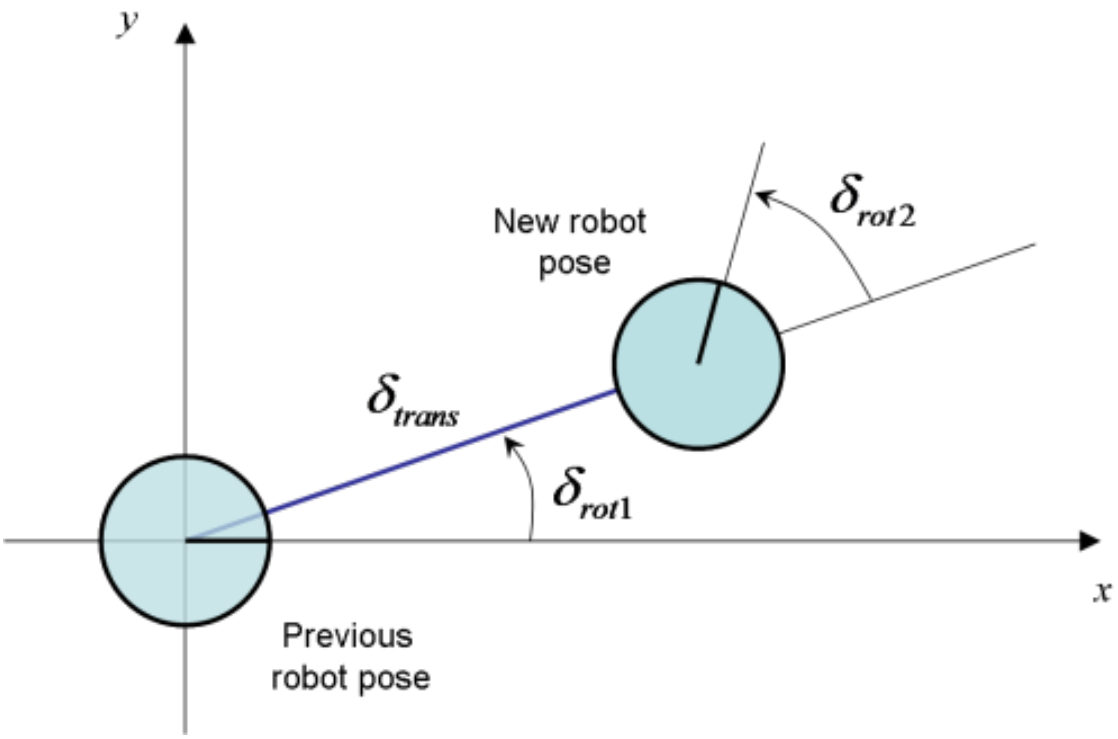

Figure 4: Odometery visualization from: `https://blog.lxsang.me/post/id/16`

The first step of SLAM is called Landmark Extraction which involves taking the laser scan input of SLAM to the system and extracting information about the environment from the scan itself. This information comes in the form of landmarks and these could be walls, people, furniture, or anything else that appears on the scan, any object and it's stored as a (x, y) position with reference to the laser scanner which is the center of the scan. These landmarks will be used later to develop the map and position estimates (see figure 6).

Landmark Extraction can be done in many ways. For many walls line extraction algorithms are used, for point data different algorithms are used. The basic idea is that we need to take the laser scan and turn the points into identifiable landmarks with positions referenced to the laser scanner (or laser frame). These landmarks are really the core of the estimation. In the next steps of SLAM we will estimate the position of these landmarks, re-read the position of these landmarks, and compare them to our estimates.

But before we get there let's talk about good and bad landmarks. A good landmark is, primarily, something that doesn't move in the map frame. This is because a landmark is a reference and if we want to figure out the new position of a robot after it has moved, this is a lot easier and more reliable if the things we are comparing the robots position to are not moving. Walls are very good landmarks. Landmarks also need to be large enough to observe. Many smaller objects or thinner objects such as poles or other thinner structures get lost in the SLAM landmark abstraction because 2D laser scanners like ours just aren't able to produce detailed enough point clouds, so SLAM thinks of these things as noise. The final thing that makes a good landmark is it must be re-observable. Landmarks need to be seen each time the robot moves in the map otherwise we lose sight of landmarks therefore

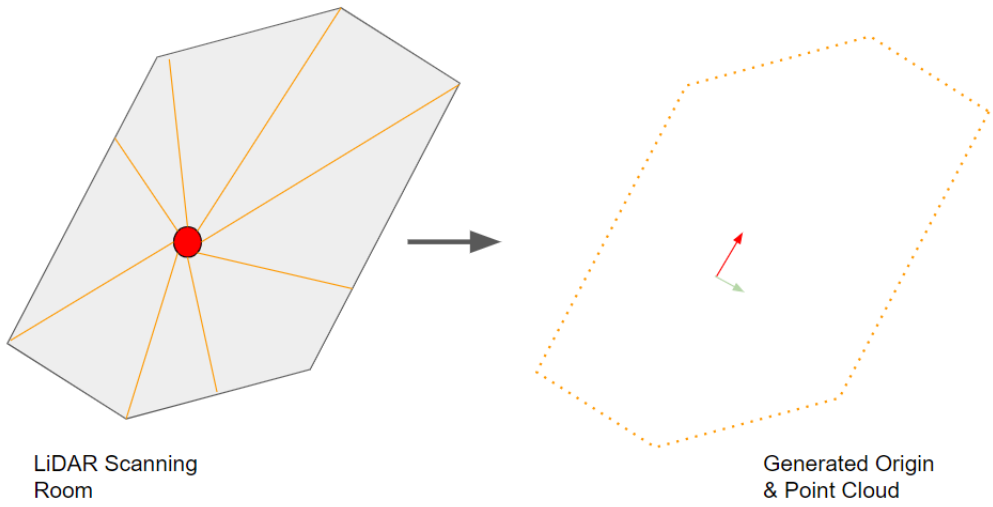

Figure 5: LIDAR generating point clouds

losing sight of our position reference. This happens as the robot moves from room to room in a building. SLAM does still work in these cases because SLAM dynamically updates the landmarks and uses the most relevant ones to estimate the robot's current position

So now, we have all of these landmarks. What comes next? This is the core of the SLAM system. The actual estimation and re-estimation. Let's say we have a current initial estimate for the robot as shown in figure 5, this usually is the position of the LIDAR in the very first scan we take even before landmark extraction happens and we take this as a given (while the algorithm does depend partially on this initial guess, it is usually accurate enough where it does not matter). We take the initial position guessed as a given just to start off the algorithm (this is similar to in Bayesian estimation how we assume the probability of all states is initially equal if you are familiar). (Figure 5,6)

Then the robot will move, if that's a turn or straight line or combination of it does not matter. The robot moves and the movement is estimated by the robot's odometery like discussed earlier. (Figure 7,8,4)

SLAM will then try to use the new estimated robot position from pure odometery to predict what the location of the landmarks should be now based on the location in the last scan. So using odometery (in our example, scan-matching odometery outlined in the figure) the robot thinks it is in a certain location compared to its old location, and it estimates where it thinks the landmarks should now be based on its movement. (Figure 8)

Finally, the algorithm will read a new laser scan, and extract landmarks from it. It will compare the location of the landmarks observed to the location of the landmarks as estimated, and try to find a happy medium between the estimation from odometery and the estimation from the landmarks (Figure 9,10).

The algorithm runs many times over, each time the robot moves, we run the above steps. But the real heart of SLAM is what happens in addition to the position estimation of the

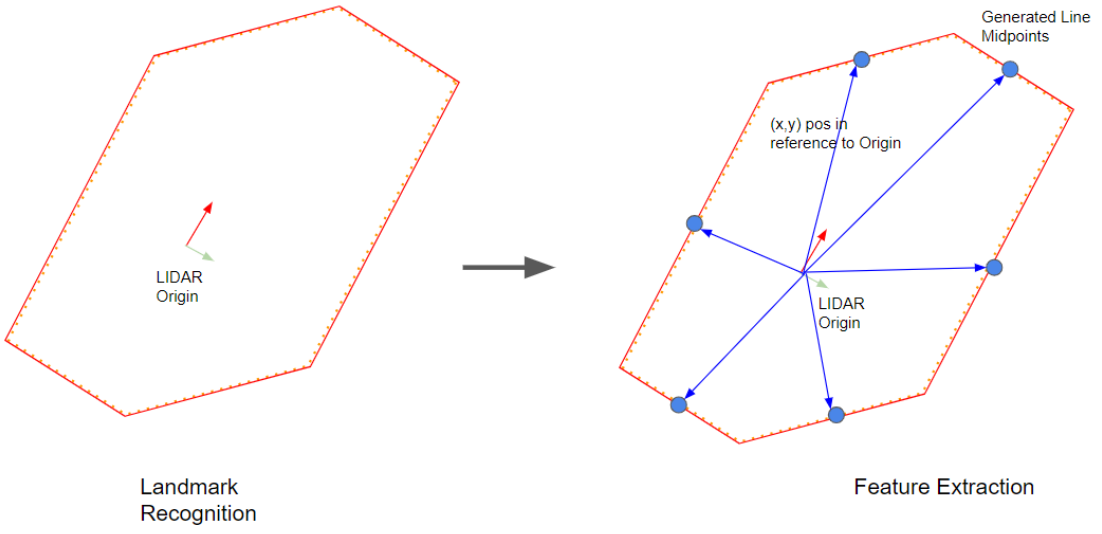

Figure 6: Landmark extraction from scan

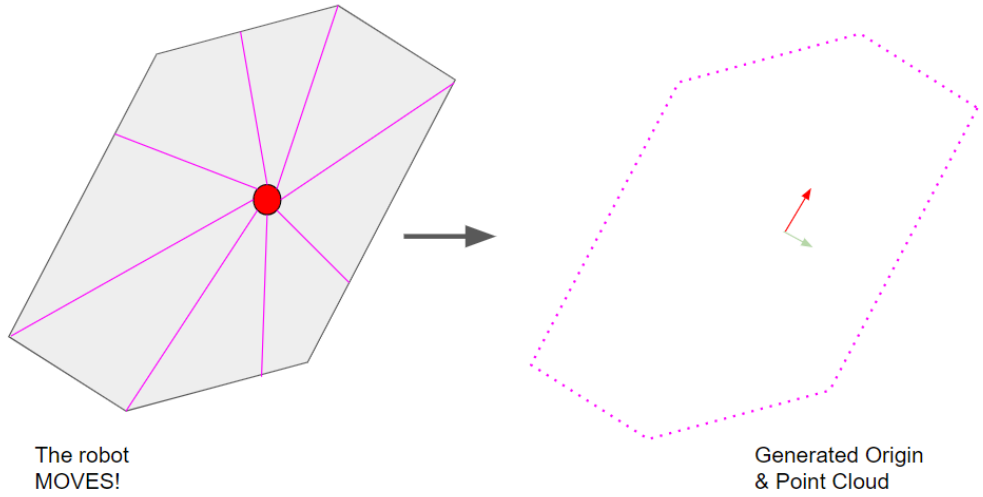

Figure 7: Generating the new scan after the robot moves

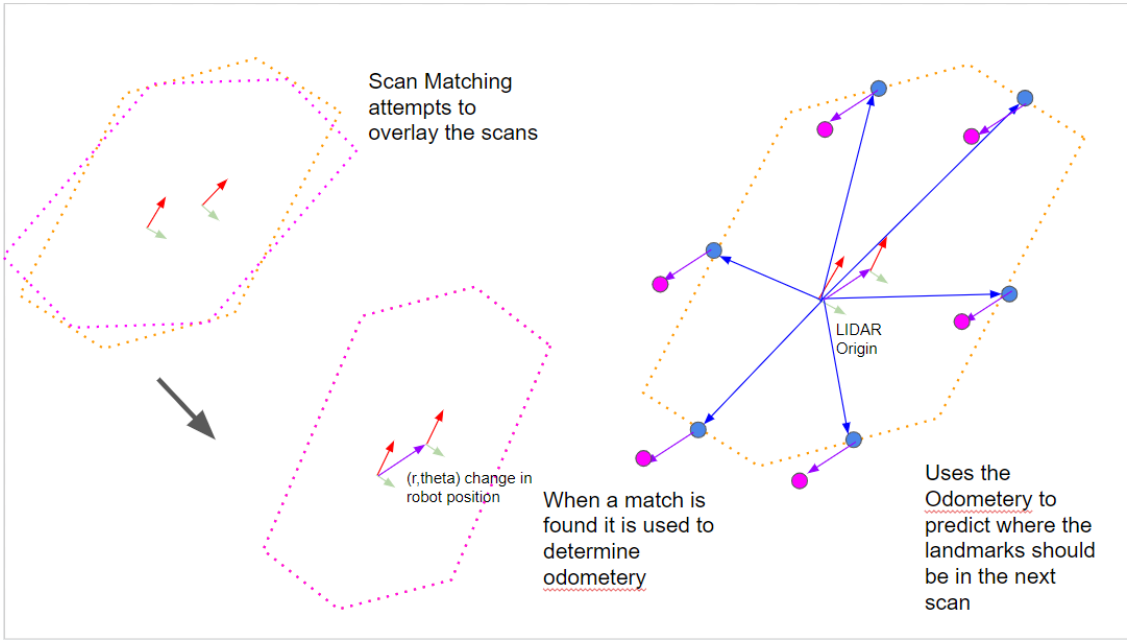

Figure 8: Estimation of landmark position based on odometery

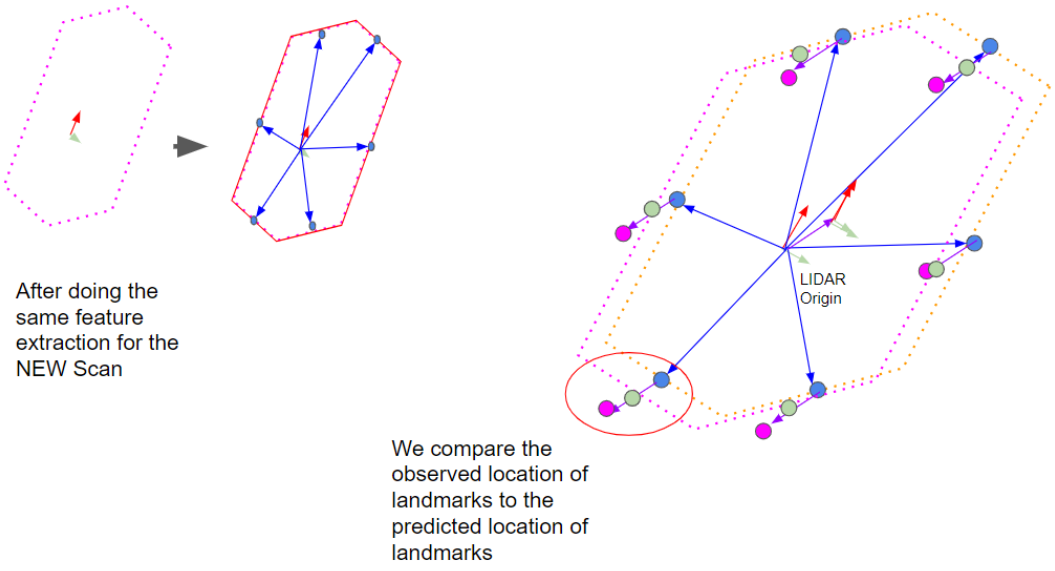

Figure 9: Comparing location of landmarks observed to estimated

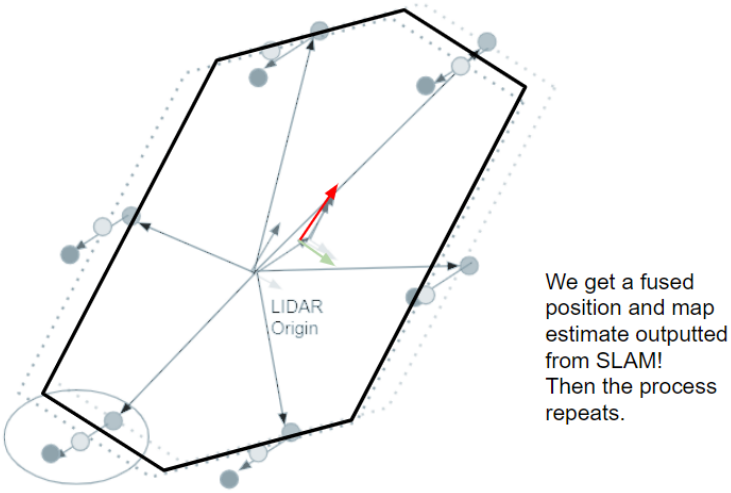

Figure 10: Generating the final map and position estimate

landmarks and the robot. SLAM is also able to recognize which landmarks are actually useful to position estimation and which ones are not. For example, SLAM will look at how the position of the robot changes with reference to landmarks and it will start to say "okay so that landmark over there keeps moving in a way I cannot predict and it really tells me nothing about where my position is so I will ignore it, as opposed to that landmark over there where I've been very accurately be able to predict its position as it seems to be a non-moving reference." The first landmark likely was a person and the second likely was a wall. SLAM can differentiate between all of these landmarks and use the landmarks that provide the most accurate position estimates as reference points. How does it do this? This gets into the implementation details of SLAM which we will briefly cover next.

4.2.3 The Matrices of SLAM (More Details)

In this next section, we will very briefly cover how SLAM does all this on a technical front. We will not go into too much detail as specifics of particle filtering and state estimation will be covered in later lectures, and there are more than enough resources online in how to implement SLAM. The goals of this section are not a how-to-implement guide but a very general, basic overview of the math behind SLAM.

From a math and algorithmic point of view, SLAM is one, giant EKF (Extended Kalman Filter), a classic for state estimation. The EKF is the core of SLAM, and it a set of matrices and mathematical computations that relate all the distinct variables associated with the system in such a way that we are able to predict their states and relate them, when we apply an EKF to estimating robotic position in unknown environments, and we include scan-based mapping, we get SLAM, that's all it is.

The EKF in SLAM is made of three matrices at the core, (1) the system state matrix X,

(2) the co-variance matrix P, and (3) the Kalman gain matrix K.

- (X) The System State Matrix: is the matrix that stores all of the information on things like position of the robot (x, y, θ), and the position of the landmarks in the global frame (x, y). This is a matrix with one column whose sole job is to store these many position variables.

- (P) The Co-variance Matrix: This is the heart of the Kalman Filter. This matrix takes every single variable, every position of every landmark and the robot position and calculates the co-variance of the robot's position, the co-variance of the landmark positions, the co-variance of the robot's position with respect to each landmark, and the co-variance of each landmark's position with respect to every other landmark. This matrix tells us how all the variables relate to each other and hence is the core of the state estimation. This is how we can relate if we observe one variable how we can predict the others. This is where the information for "predict where the landmarks will be based on odometery estimates" comes from.

- (K) The Kalman Gain: this matrix is the final product of the above matrices and calculations resulting from them. These are the system gains. It uses the co-variance matrix to understand how the variables relate to each other and at each time step updates a "gain" for each of the landmarks and the odometery which tells SLAM how much to trust each landmark in estimation and how much to trust odometery. This is where the information from the "compare estimated landmark positions to observed ones." This is where the real learning happens where SLAM learns which landmarks to trust and which ones to not use at all in position estimation.

We will not be going through the calculations behind all of this as the math isn't complicated but long and time consuming and will be covered in later lectures. We wanted you to get a sense that everything SLAM does is possible through three very simple matrices, that being said it really is a very simple algorithm for the most part. The real innovation in this algorithm isn't necessarily HOW it does estimation but how it creates the inputs for estimation and the process/steps it goes through. That being said, SLAM does in fact have some limitations.

4.2.4 SLAM Limitations

SLAM with scan-matching odometery specifically has many limitations we will briefly cover in the hopes that you, with your own implementations, may be able to fix some of them! When our lab staff tested this implementation of SLAM we found some inherent issues that we will now outline.

- Limitations of Scan Matching Odometery: Scan matching odometery rides on one major assumption, the scan inputted does not change significantly from time-step to time-step and that because of this the system will easily be able to match the scans. What this means is sometimes under specific operation conditions these rules are broken and SLAM freaks out completely. For example, if I have a room with five walls and I

start running SLAM and get a map of these five walls and then randomly add a large, wide obstacle (diving wall or similar) in the middle. The map does not visibly update the existence of this new obstacle unless you start to drive the robot around. What's happening here is when you first insert the wall, SLAM saw absolutely no reason for this massive unexpected change in scan and it's likely unable to match the radically different scan with the old one. So it simply ignores it and attributes it to sensor failure or error. But if you start to move the robot around, then SLAM starts relying on the most current information, matching the new scans with the scan with the new wall and is able to re-generate a new map. This will likely not occur in our use-case but is an interesting predicament. (Hint: this can be solved with regular drive-based odometery, think about why!)

- Limitations of Laser Scanners and Input Devices: Laser scanners have a certain refresh rate and spin speed - right? This isn't usually a problem because these speeds are fast, much faster than how fast a robot is supposed to spin or turn in our use case. But sometimes when the rotational velocity of the robot becomes fast (not close to the spin speed of the LIDAR by any means, but significant compared to its spin speed), what tends to happen is the scan itself will become highly distorted producing strange curved lines, random hallways and paths on the map resulting from a scan generated while rotating quickly and also SLAM's inability to match this scan to the previous scans.

NOTE: The origin of the SLAM reference-frame is usually the location of the LIDAR in the first scan inputted at time-step t=0, and everything including the landmarks at the initial time-step is with reference to that. I.e. the starting location of the robot is the origin of the map-frame.

5 Hector SLAM

`https://wiki.ros.org/hector_slam` `https://github.com/NickL77/RPLidar_Hector_SLAM`

Hector SLAM is ROS package that implements SLAM. This implementation can use, but does not require odometery.

6 MASLAB & LIDAR

Historically, MASLAB robots have not used LIDAR sensors prior to 2020. In the past, the same distance information could be measured using either an array of ultrasonic distance sensors for long range and infrared proximity sensors for short range. There are several advantages LIDAR provides over both ultrasonic and infrared sensors of those types. Most noticeably a LIDAR sensor fills the role that all of these sensors played in a single easy to use package. The LIDAR works for both long and short range distances. Additionally, LIDAR is also less susceptible to interference that is common for ultrasonic sensors.

7 Read More (SLAM References)

- SLAM THEORY: http://courses.csail.mit.edu/6.141/spring2013/pub/lectures/Lec16-SLAM.pdf
- SLAM BASICS & IMPLEMENTATION: https://dspace.mit.edu/bitstream/handle/1721.1/119149/16-412j-spring-2005/contents/projects/1aslam_blas_repo.pdf

References