Autonomous Development Team

Recruitment Task

- This task is inspired by this paper. We tried to explain it more and make things clearer
- If you don't understand anything, search. If things are still unclear, you can reach to us
- We know it's not an easy task, but we are challenging you!
- Solve it step by step, don't expect it to be clear from the first time
- You don't have to stick to our steps if you can reach the same result
- You must tell us how to run your code
- Attach images if you want to show us anything

Introduction

In this task you will be applying a **segmentation algorithm** applied to a two-dimensional point-cloud extracted by a LIDAR device.

LIDAR is a remote sensing technology that allows **measuring distances** by emitting a laser beam and analyzing light reflected by the object.

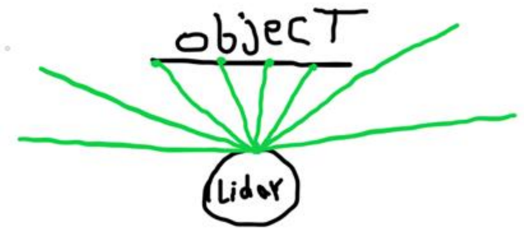


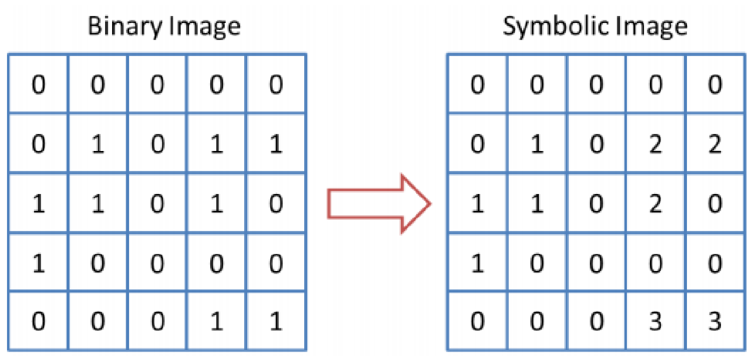
Figure 1 Lidar Readings

LIDAR data segmentation is a necessary part in many tasks such as *obstacle detection* and *object detection/recognition*.

You will be using a **connected components algorithm** to do the segmentation.

Brief

One of the most essential algorithms in *pattern recognition* and *image processing* is the **Connected Components (CC)** algorithm. The purpose of the CC algorithm is to transform the input *binary image*, into a *symbolic image* where each connected component will have a unique label.



Binary Image

0	0	0	0	0
0	1	0	1	1
1	1	0	1	0
1	0	0	0	0
0	0	0	1	1

Symbolic Image

0	0	0	0	0
0	1	0	2	2
1	1	0	2	0
1	0	0	0	0
0	0	0	3	3

Figure 2 The left part of the figure shows a binary image and the right side shows a result of labeling process using CC

We extend the CC algorithm from labeling binary images to **segmentation of LIDAR data**.

Input

Lidar readings (scans):

An array of floats each representing a scan ray (which has a magnitude and a direction) the float **value** stored in the array represents the **magnitude** and its **order** in the array represents the **direction** (angle). Note that if a lidar scan ray **didn't hit** an obstacle the magnitude of the ray will be **zero**.

The angle between any two consecutive readings (delta) equals $\mathbf{1}°$, also the range of angles is $[\mathbf{0} \rightarrow \boldsymbol{\pi}]$. That means that the input array has total of $\mathbf{180}$ **readings**.

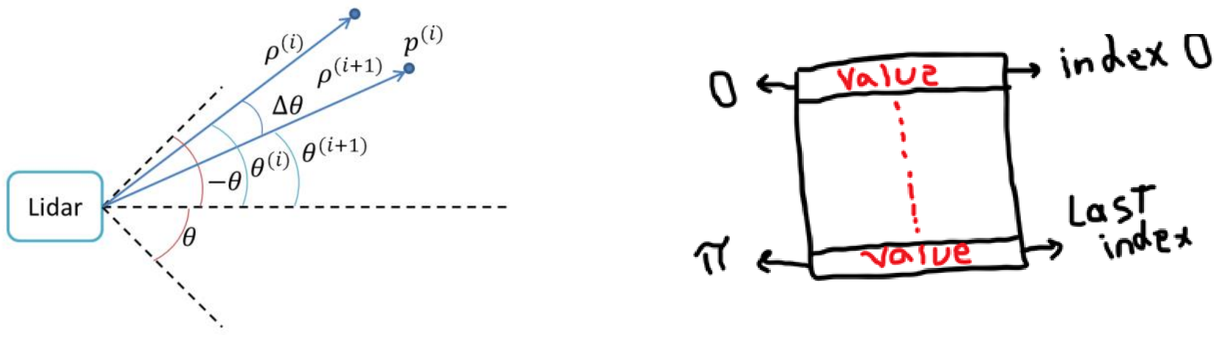


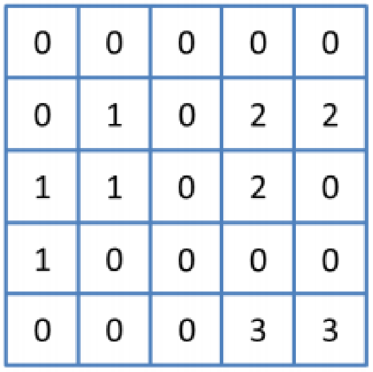
Figure 3 Example of lidar scans input

***Lidar scans input should be read from a file called "scans.txt"**

Output

Labelled Grid Cells:

A grid of cells where each cell has a label (like symbolic image seen Figure 2).



0	0	0	0	0
0	1	0	2	2
1	1	0	2	0
1	0	0	0	0
0	0	0	3	3

Figure 4 Example of labeled grid cells

***Labelled cells output should be stored to a file called "cells.txt", where each row in a line and values are space separated**

Definitions

These definitions may help you if you sticked to paper's method

```
// Holds information about a discretized point
struct DiscretePoint
{
    int x;       // Discretized x-coordinate of point
    int y;       // Discretized y-coordinate of point
    bool marked; // Was the point labelled before
};
```

```
// Holds information about a grid point
struct GridCell
{
    int label;            // Label given to cell and its linked points
    list<Point *> points; // Points inside cell and linked to it
    bool activated;       // Was this cell labelled before
};
```

```
// Holds information about a labelled point
struct LabelPoint
{
    int x;     // Discretized x-coordinate of point
    int y;     // Discretized y-coordinate of point
    int label; // Label given to point and its linked cell
};
```

Pseudo Code

The whole process is summarized as follows:

1. Convert all the input points $P_i = (\rho_i, \theta_i)$ (originally in polar coordinates) to points $P_c = (x_i, y_i)$ in Cartesian coordinates
2. Discretize the obtained Cartesian values
3. Project all the 2D points onto an occupancy grid
4. Find connected components over the grid
5. Return labels

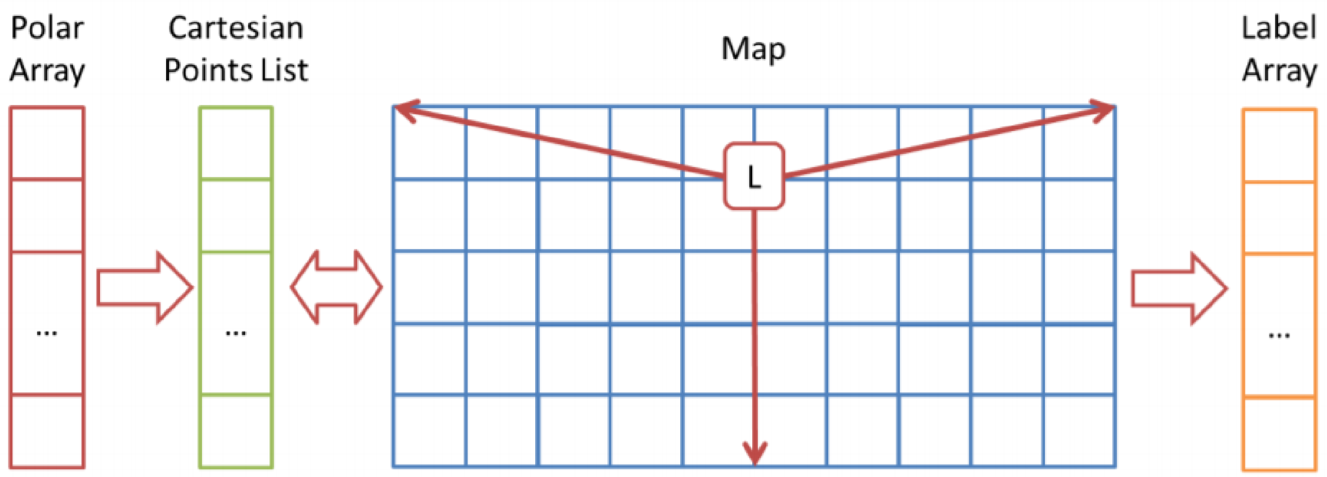


Figure 5 Structures used in algorithm

Preprocessing Stage

Before applying the CC algorithm, the points should be converted to Cartesian coordinate system.

To convert from polar to cartesian coordinates, this formula is used:

$$x_i = \rho_i \cos\theta_i$$

$$y_i = \rho_i \sin\theta_i$$

We assume each cell in grid is $\mathbf{1m} \times \mathbf{1m}$ to simplify, so discretization is just ignoring fractions.

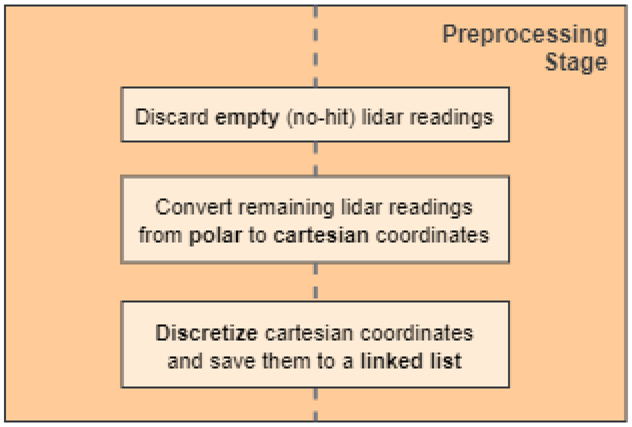

Preprocessing Stage
Discard empty (no-hit) lidar readings
Convert remaining lidar readings from polar to cartesian coordinates
Discretize cartesian coordinates and save them to a linked list

Figure 6 Preprocessing stage flow chart

Dimensions of the grid are calculated as follows:

$$width = 2 \times lidar\_max\_range$$

$$height = lidar\_max\_range$$

Assume: $\boldsymbol{lidar\_max\_range = 20m}$

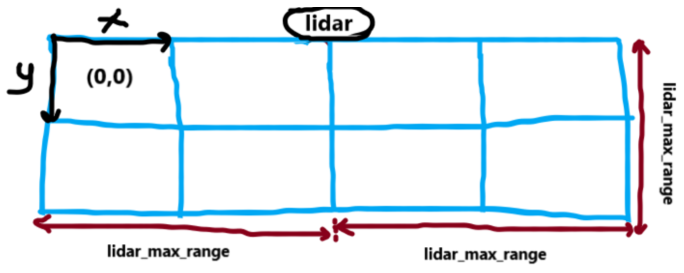


Figure 7 Grid cells dimensions and details

Filling & Linking Stage

Each cell can be thought of as a space of a real world with a constant size that can have 0 or k points of the scan. By putting all together, the grid is treated as a binary image, where we can consider each cell as a pixel that can have two values, 0 or 1 depending on if it has points inside or not.

Then points are projected onto occupancy grid. This grid is treated as a binary image and used afterwards for labeling.

All the points projected inside a grid cell are linked to that cell (which will be used for labelling). Linking is done using pointers or indexes.

The process of filling up of the occupancy grid given by:

$$x = x_i + l_x \qquad l_x = lidar\_max\_range$$

$$y = y_i + l_y \qquad l_y = 0$$

l_x, l_y are the x and y coordinates where the LIDAR is placed on the map. This is just to consider lidar place and put points in their right locations in grid.

Connected Components

You choose a point from the discretized points and remove it from list, then you get its associated grid cell using the link between them (the pointer or the index).

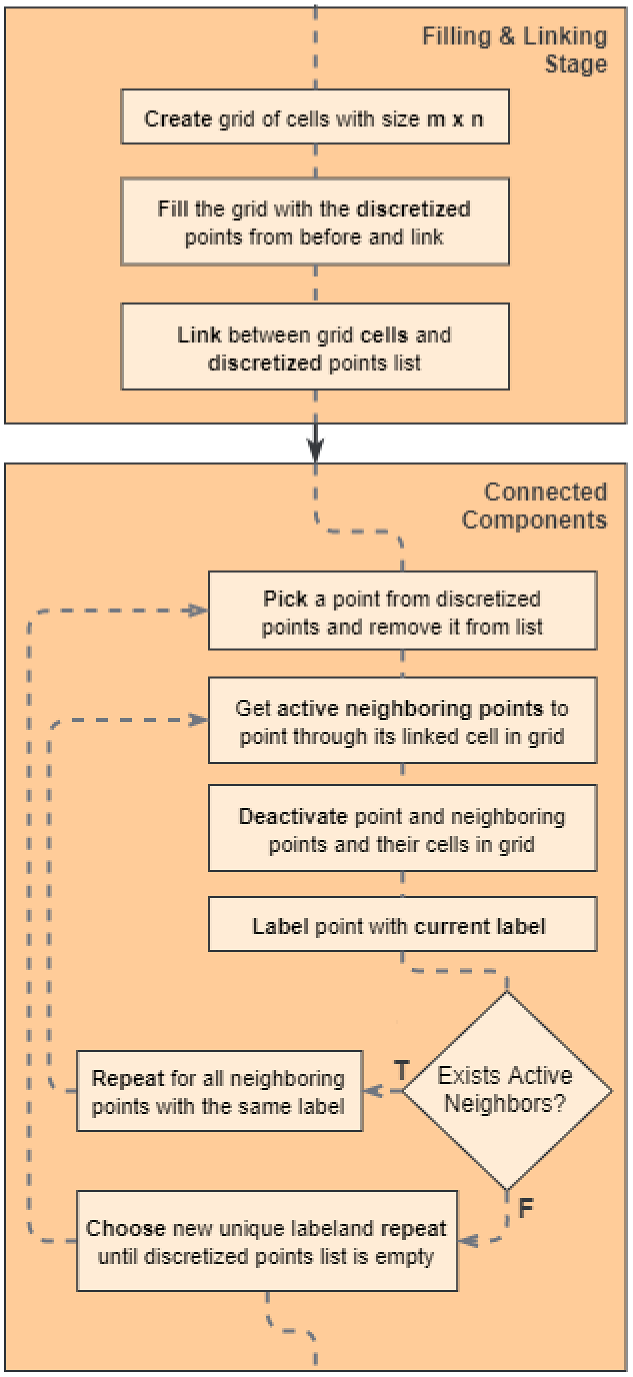

Filling & Linking Stage
Create grid of cells with size m x n
Fill the grid with the discretized points from before and link
Link between grid cells and discretized points list
Connected Components
Pick a point from discretized points and remove it from list
Get active neighboring points to point through its linked cell in grid
Deactivate point and neighboring points and their cells in grid
Label point with current label
Exists Active Neighbors?
T
Repeat for all neighboring points with the same label
F
Choose new unique labeland repeat until discretized points list is empty

Figure 8 Linking and Connected Components flow chart

Then you get all the 8-connection cells around this cell, then you get all the points that fall in all these cells, all these points are the previously chosen point neighbors, so you iterate over them and do the same again (also get their neighbors).

All these points belong to the same object and must be labeled with one unique label. When you finish, you choose another point from the list and start again.

You finish when you have labeled all points and don't forget to remove or mark the point you have labeled to avoid labeling it again.

Another Method for Connected Components: Search for *Connected Components* in image processing. You don't have to stick to paper's method, but it will be better if you do.