Report of Assignment-2

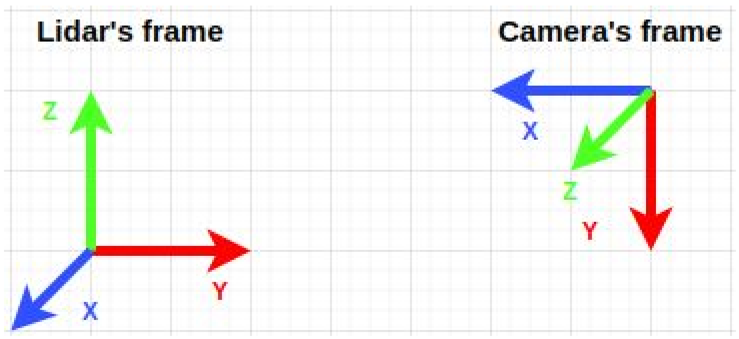
Avinash Prabhu- 2018102027
Rahul Swayampakula- 2018102037

Task 1- Point Cloud Registration

Summary of steps to get the final Point Cloud Representation

1. Transform the point cloud from the **Lidar's frame** to the **Camera's frame**.
2. Apply the global transformation obtained from the .txt file.
3. Transform the point cloud from the **Camera's frame** to the **Car's frame**.

1. Transform the point cloud from the Lidar's frame to the Camera's frame.



- The LiDAR points we obtain from the bin files are represented with respect to the **Lidar's frame**.

- On the other hand, the global poses we obtain are with respect to the **Camera's frame**.

- Thus, in order to accurately transform the point clouds, we must first transform them from the **Lidar's frame** to the **Camera's frame**.

- We can do so with the help of a rotation matrix.

Calculating the rotation matrix to convert from Lidar's Frame to Camera's Frame

- Let X_C , Y_C , Z_C be the axes of the **Camera's frame**.
-
- Let X_L , Y_L , Z_L be the axes of the **Lidar's frame**.
-
- The rotation matrix from **Lidar's frame** to the **Camera's** frame is given by R_L^C.

$$R_L^C = \begin{bmatrix} X_L^C & Y_L^C & Z_L^C \end{bmatrix} = \begin{bmatrix} X_L \cdot X_C & Y_L \cdot X_C & Z_L \cdot X_C \\ X_L \cdot Y_C & Y_L \cdot Y_C & Z_L \cdot Y_C \\ X_L \cdot Z_C & Y_L \cdot Z_C & Z_L \cdot Z_C \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}$$

- We then append [0 0 0] as the 4th row and [0 0 0 1] as the 4th column to accommodate for the matching the dimensions of multiplication in the next step.

The final rotation matrix to transform from the Lidar's frame to the Camera's frame represented in our code like this-

```
cam_trans = np.array([[0,-1,0,0],[0,0,-1,0],[1,0,0,0],[0,0,0,1]])
```

Finally, we transform the points like this-

```
cam_trans@read_pc[i]
```

Where-

1. cam_trans is the rotation matrix defined above.
2. read_pc[i] is the ith point cloud we are transforming.

2. Apply the global transformation (for initial camera frame) obtained from the .txt file.

- The next step is to apply the global transformation.

```
np.resize(global_trans[i],(3,4))@cam_trans@read_pc[i]
```

- Here-
    1. global_trans[i] is the global pose (pose w.r.t initial camera pose) of the ith point cloud.
    2. We resize it to a matrix of size 3x4 for multiplying it with the point cloud.

3.Transform the point cloud from the Camera's frame to the Car's frame

Steps to convert from Camera's Frame to Car's Frame

- The **Car's Frame** is the same as **Lidar's Frame**, so let us refer to the **Car's Frame** as the **Lidar's Frame.**

- So, we need to find the rotation matrix from **Camera's frame** to **Lidar's Frame** which is the transpose of the R_L^C matrix calculated in **Step 1**.

- The matrix given by-

$$R_C^L = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

- The final rotation matrix to transform from the **Camera's frame** to the **Car's frame** represented in our code like this-

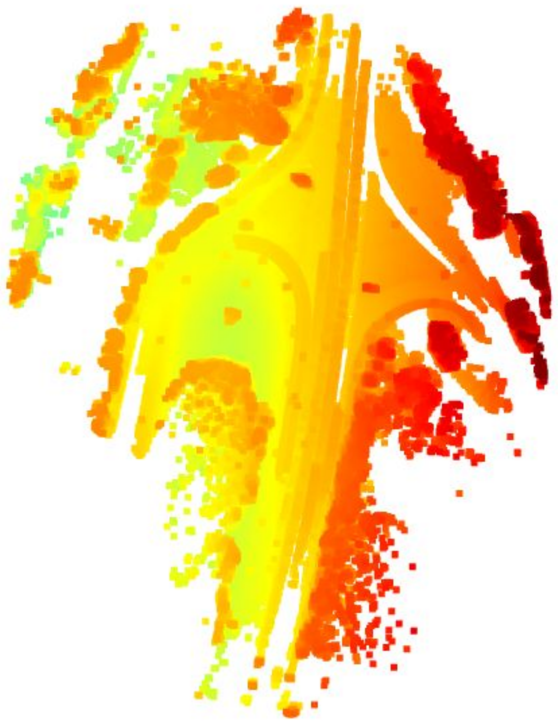
```
back_trans = np.array([[0,0,1],[-1,0,0],[0,-1,0]])
```

- Finally we transform the point cloud to the Car's frame is code-

```
(back_trans@(np.resize(global_trans[i],(3,4))@cam_trans@read_pc[i]))
```

Once we complete the 3 steps mentioned above for all the point clouds, we can represent them simultaneously. They are now an accurate reconstruction of LiDAR scans

Output for 2.1:



Task 2- Occupancy grid Construction

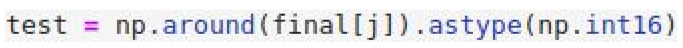
Summary of steps to get the final Occupancy grid

1. Create an empty occupancy grid.
2. Eliminate Duplicate Points.
3. For a given $[x, y]$ coordinate, count how many z values exist.
4. Remove the points that do not have enough z values.

1. Create an empty occupancy grid

- Take an array of points that we constructed in the **Task1**.

- Since the occupancy grid is basically an image with discrete integer indices, we will round off the Lidar's measurements to integer values.
- Here is how we implemented it in code-



```
test = np.around(final[j]).astype(np.int16)
```

Where

- final[j] is the jth point cloud.

- Next, we find the max and min values for the x and y coordinates in order to determine the dimensions of the occupancy grid.

  - $x_{min} \leq x_{co} \leq x_{max}$
  - $y_{min} \leq y_{co} \leq y_{max}$

Where

- x_{min} is the minimum of all x- coordinates in the array.

- x_{max} is the maximum of all x- coordinates in the array.

- y_{min} is the minimum of all y- coordinates in the array.

- y_{max} is the maximum of all y- coordinates in the array.

- x_{co} is the x- coordinate.

- y_{co} is the y-coordinate.

Here is how we implemented the above points in code-

```
x_y = test[:,0:2] # Extracting the x and y coordinates

# Getting the max and min values along the x and y coordinates
max_x = np.amax(x_y[:,0])
max_y = np.amax(x_y[:,1])
min_x = np.amin(x_y[:,0])
min_y = np.amin(x_y[:,1])

len_x = max_x - min_x
len_y = max_y - min_y
```

- Once we have the dimensions of the occupancy grid, we create it with dimensions $[(y_{max} - y_{min}) + 1, (x_{max} - x_{min}) + 1]$ initialised with zeros.

Here is how we implemented the above point in code-

```
#Creating the occupancy grid and initialising it with 0s
occupancy_grid = np.zeros((abs(max_y-min_y)+1,abs(max_x-min_x)+1))
```

- The dimensions are in this way because to visualise the Y-axis in the vertical direction and X-axis in horizontal direction.
- Where block [y-y_min,x-x_min] corresponding to point (x,y)

2. Eliminate Duplicate Points

- Since we want to generate different values of z for a particular (x,y), we need to remove the duplicate $[x, y, z]$ coordinates.

  Here is how we implemented the above point in code-

```
# Eliminating any duplicates of points
xy_u = np.unique(test,axis=0)
```

3. For a given $[x, y]$ coordinate, count how many z values exist.

- After removing duplicates let's count the occurrence of $[x, y]$ in the array which is nothing but counting the different values of z for a pair $[x, y]$.

  Here is how we implemented the above point in code-

```
# Mapping each point in the point cloud to a pixel in the occupancy grid
for i in range(len(xy_u)):
    x = xy_u[i,0]
    y = xy_u[i,1]
    occupancy_grid[y-min_y,x-min_x]+=1
```

4. Remove the points that do not have enough z values.

- After getting the count of multiple z we mark the cell as an obstacle only if the count of z values for that cell is greater than threshold else it will be considered as empty.

  Here is how we implemented the above point in code-

```
# Removing the points that do not have enough z values
occupancy_grid = (occupancy_grid>threshold)
```

- The results are plotted using the matplotlib library.
- The results are saved using the PIL library.

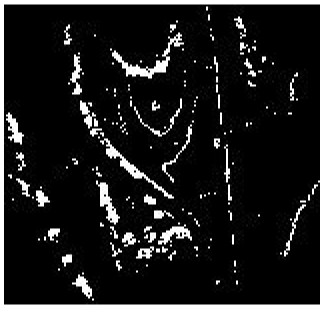
In the tasks of 2.2 (a) and (b) the major differences are:

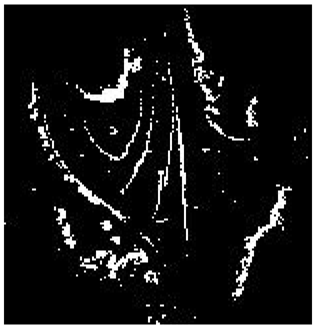
1. In 2.2(a) the array we took is only points corresponding to one timestamp calculation whereas in 2.2(b) we took the array of points including different time stamps.
2. The threshold taken in 2.2(a) case is 1 whereas 2.2(b) we will increase the threshold for better estimate of the obstacle at that particular position. This is because to get a better probabilistic value from noisy measurements.

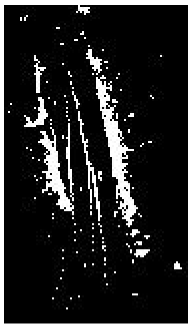
Output for 2.2:

Note: In the generated occupancy grid the Global X-axis is in Horizontal direction.

Part-a: Example of outputs for occupancy grids generated for each time stamp. (Threshold =1)

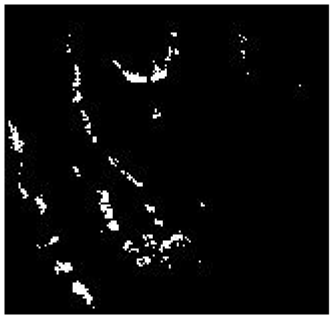


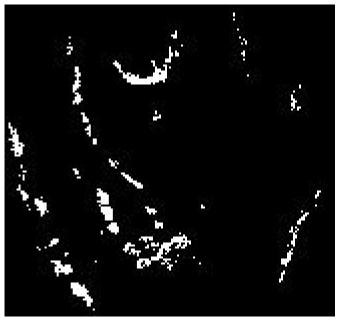


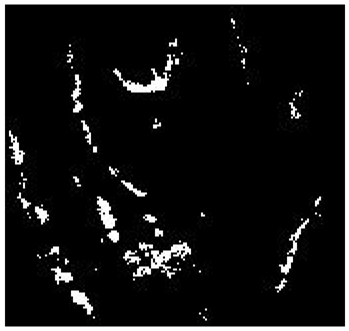


Bin 0 **Bin 20** **Bin 76**

Part-b: Results of 5,10,15 scans respectively from left to right with threshold = 3.







Observations:

- While doing the assignment, we saw how the lidar observations, frame transformations are helping in building the actual scenario of the environment which will be useful for car to travel through the environment.
- When the observations are taken from LIDAR all the points are with respective sensor frame but the pose of the car is estimated by ICP which is in camera frame. So we get pose information of car in camera frame only (i.e. Matrix in 01.txt). Either we should convert the transformation to LIDAR frame or points to the camera frame to get the correct set of points.
- Here we converted points to camera frame and performed the transformations.
- Now we will get all the points with respective the initial camera frame i.e. it is the view of the camera at the initial position. This is converted to global frame to get correct estimates and the PCD generated gives an idea of distance of the points from the start.
- And if we observe the PCD there is a patch lo line which can be differentiable from the rest. That is the one which is showing the free path

- In real scenarios the sensors will have some error so in order to get a better estimate of obstacles around we try to take multiple samples at different timestamps and construct the original environment with better estimates.
- This was observed by taking single and multiple point clouds .The use case of occupancy grid with occupied and unoccupied measures were seen in this and also helps in locating the obstacles around the car to avoid collision.

Interpretation of the point clouds:

From the obtained point clouds and occupancy grid construction we may say that initially the car is moving towards the front by0 some distance and then taking a turn, we can be sure that there are no obstacles in the path by observing the outputs of PCD and occupancy grid.

<u>Results:</u>

Please find complete results here

Results folder contains 2 folders 2_1 and 2_2
Results/2_1 contains .pcd and .png file of the obtained pcd
Results/2_2/1st/photos → individual outputs
Results/2_2/1st/ → contains a timelapse video of all photos in photos folder
Results/2_2/2nd/ → outputs of multiple scans

<u>Work Division:</u>

- The work was divided equally to reduce the workload on each other and to complete tasks on time.
- The theoretical parts were solved by both of us and cross checked each other.
- For the coding part the 2.1 was coded by Rahul and 2.2 by Avinash.
- Report and README were written by both of us.