Assignment 2 - Lidar

F110-2020

Jay Patravali (patravaj@oregonstate.edu)

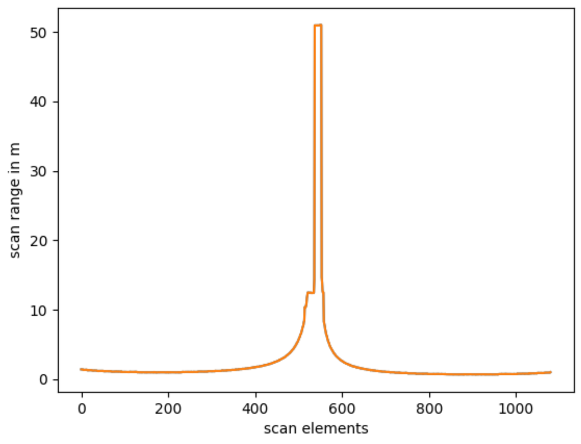
16 April 2020

Overview and Description

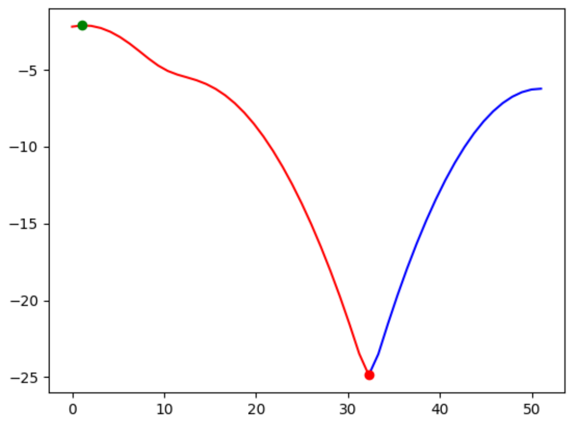
This assignment is on gap finding from laser scans. I tried 3 techniques

1. Kmeans clustering
2. Kernel Density Estimation for clustering
3. Simple Geometry based solution

I tried experimenting with K-means clustering however the biggest problem with it is manually selection the number of clusters which is hard to tune. Secondly, K-means is not recommended for 1-dimensional range data : https://stackoverflow.com/questions/11513484/1d-number-array-clustering So, then I tried Kernel Density Estimation for clustering range data. It did seem promisining in the beginning as it was able to succesfully split the data into walls and gap as shown in Fig 1. In Fig 1 (a) is plot of 1080 range values. Fig 2 (a) is applying KDE and obtaining the log-probability density of range values. The red dot is where the data is splitted into walls and the gap for robot initial position. The problem with KDE clustering is that it gives a common label to all points within a same range. For example, the robot's left and right side could have walls with equal range values. KDE will cluster all these into a single label. So its hard to distinguish between different walls and gaps(high value ranges).



(a) Plotting 1080 Range values. Peak is where the gap values reach max_range



(b) log probability density under the model.

Figure 1: KDE computation for range data obtained when the robot is spawned.

Finally, I stuck to a simple method of finding a gap due to spike in range values. Then using cosine rule it was possible to obtain the gap centers. This I have explained with Figure below.

0.1 Algorithm Pseudocode for gap finding

```
1. scan_callback(): Collect scan message from ROS subscriber
2. find_gaps(): Function to compute gaps and return best_gap_center and all gap_info
   (a) Filtering: clip range_data to max and min range
   (b) find gap segments in range data. Loop idx=0.. len(range)
       i. delta = difference between consecutive range data – data_arr[idx+1] - data_arr[idx]
       ii. if delta is greater than permissible difference then add that idx to a list – segment_storage
   (c) compute gap center and gap length. Loop i=0...len(segment_storage)
```

        i. angle subtended by gap − scan_gap_angle = angle_increment *(segment_storage[i+1]-segment_storage[i])
        ii. length of side 1 of the gap triangle s1 = data_arr[segment_storage[i]]
        iii. length of side 2 of the gap triangle s1 = data_arr[segment_storage[i]]
        iv. given: scan_gap_angle, s1, s2 we compute all the parameters of the gap triangle using cosine rule (see figure 2).

    (d) Once we have r_vector and angle from the robot's origin frame of reference to the center of the gap segment − central_angle. we need to transform the gap center to robot's local reference frame.

    (e) return gap centers, gap lengths, and number of gaps

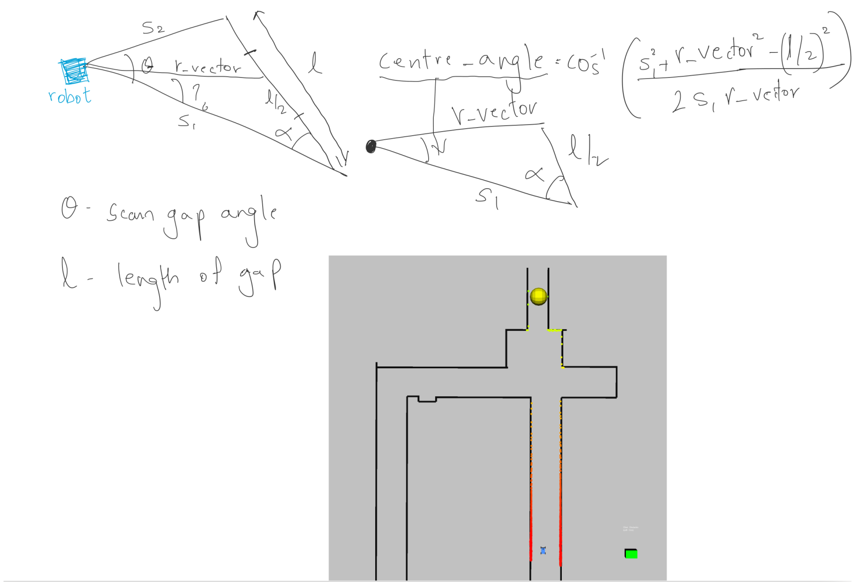
3. best_gap = max(all gap lengths)

4. publish gap centers and gap info messages

0.2 Algorithm Explanation for gap finding

Before finding the gaps, we need to ensure that the data range array is free of NaN and range values clipped within set limits. The algorithm I ended up using is pretty simple and is based on a simple heuristic that the difference between adjacent range values for a wall or continuous object reflection will be within reasonable value. The gap can be identified if the diff is greater than a threshold. This threshold cannot be same for high and low range values: for example a wall close to robot will have range values changing by small amount 0.1-0.3m. However, walls far away from the robot will have range value differing by 0.5 - 1.5m. To prevent false-separation by having a single fixed threshold value, we can instead assume a threshold that scales with the magnitude of the range value which I refer in my code as scale_permissible_diff. Once we have different segment indexes as starting and end range values for a gap, we can then compute the gap center. See Figure below for the gap triangle variables. Using cosine rule, we can locate the gap center and transform its polar coordinates to cartesian in the robot frame of reference. For transformation we need to add the minimum scan angle + angle subtended by gap center + starting angle of the gap segment(angle_increments*start_index). Finally, we publish the gap_center with largest length on 'gap_center' topic. Additionally, a custom message 'gap' is published which contains number of gaps, all gap centers in Vector 3 format and their lengths in Float64 format.

Here, the parameter scale_permissible_diff I have set as default as 0.5 after tuning. Reducing it will add more gap cluster, sometimes same gap centers are repeated. In a way its loosening constraints on gap segmentation and increasing the number of gaps. Additional filtering maybe required to remove repetitions in gap centers. Additionally, what could have been done is storing these gap centers in queue for 5 time steps so it would smoothen the gap position visualization. Currently, the gap center marker seems to be jumping in small neighborhood because of small changes in its coordinates.



robot
s_2
θ
r_vector
$?_b$
s_1
l
$l/2$
α
centre_angle $= \cos^{-1}\left(\frac{s_1^2 + r\_vector^2 - (l/2)^2}{2 s_1 r\_vector}\right)$
r_vector
$l/2$
α
s_1
θ - Scan gap angle
l - length of gap

Figure 2: Geometry calculations. Yellow Sphere Rviz marker is the gap center.