LiDAR Object Tracking

Git: <https://github.com/gayan68/LiDAR_Tracking/tree/main>

The input for this program should be a list of boundary boxes with 7 parameters for each boundary box ([x,y,z for center], [x,y,z of the extent], yaw).

The program has 2 parts.

1. Track the objects based on tree parameters
   * Track the closest distance between frames in X, Y 2D space (z: height is ignored).
   * Use Kalman filter to predict the next position of a boundary box in the frame. This improved the performance of the algorithm.
   * Compare velocities of the boundary boxes between consecutive frames to reduce the error of assigning the Tracking ID.’

The output of this program is json file (“3d\_ann.json”)

1. The second program is a modified version of “run\_demo\_lidar\_bev.py” by cadc\_toolkit (<https://github.com/mpitropov/cadc_devkit>)

This program generates the birds eye view video/GIF based on the criteria selection.

## Kalman filter

The Kalman filter can estimate the next location almost perfectly. However, this also leads to ghost object in the view. One way to reduce ghost objects is reducing the countdown timer (settings.yaml).

Below is an example case

Real-Position: [15.734694468084708, -33.064940551267085, 0.7582789183589035]  
Pred-Position: [15.395614800107332, -34.23481291370683, 0.9988193761537925]

Real-Position: [15.734694468084708, -33.064940551267085, 0.7582789183589035]  
Pred-Position: [15.395614800107332, -34.23481291370683, 0.9988193761537925]

Real-Position: [42.185583588899014, 49.2342083264598, -2.1252457553323834]  
Pred-Position: [39.691446207640055, 50.12455299680165, -2.8269616590869115]

## Compare velocities

Taking the velocity of the object in to consideration can reduce false positive cases. For example, maybe a false positive object could be the closest one to the target. In normal case the tracker would pick the closest object in the next frame to the predictive position. However when the velocity is taken into the consideration, before selecting the best object the velocity also checked. However there is a threshold for the velocity. Therefore the velocity should not be much exactly but should be within the threshold. This could improve the performance as shown in below Figure 1and Figure 2. The Red square highlights the improvements of Figure 2 over Figure 1. Also, I identified the best velocity threshold is 50 where it can then differentiate between two close tracking objects with significant close velocities as shown in Figure 2 and Figure 3 in Green color.

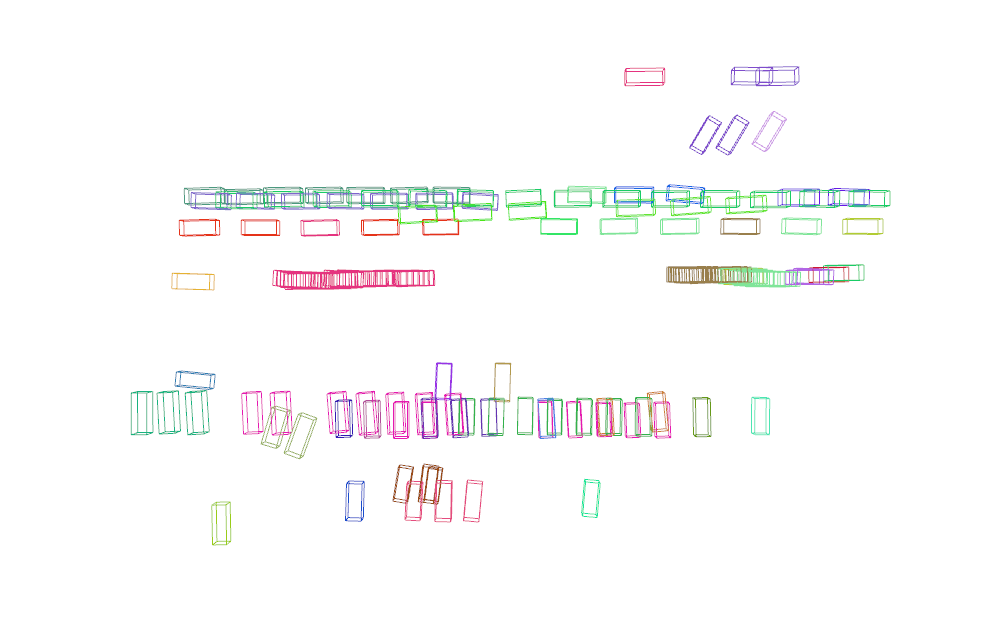


Figure 1: Object tracking for 5 boxes for 30 frames. No velocity-based tracking.

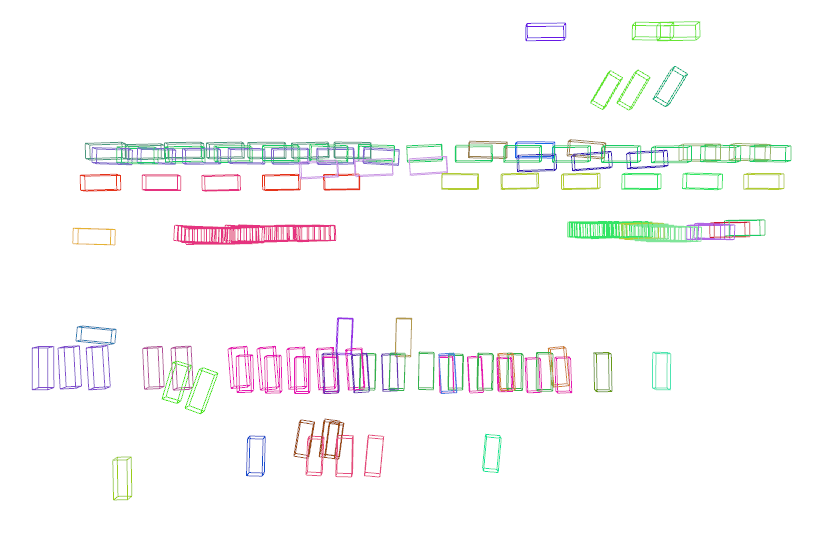


Figure 2: Object tracking for 5 boxes for 30 frames. Velocity-based tracking with velocity threshold of 100.

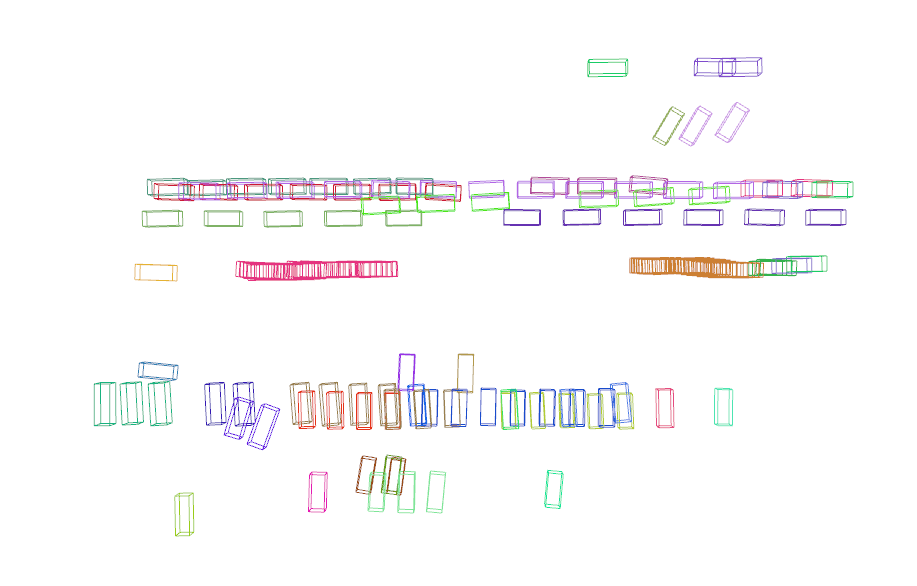


Figure 3: Object tracking for 5 boxes for 30 frames. Velocity-based tracking with velocity threshold of 50.

# How the program works

File: object\_traking.py

In the 0th frame, the program initialized trackers (generate\_traking\_id()) for all boundary boxes. This includes assigning a new tracking ID initializer kalman filter tracker appending to vehicle\_ids list.

During the 1st frame onwards, the algorithm loop through each boundary box in the frame, then again loop through each tracking ID to find the best match based on closest and velocity match. However, there are thresholds for the closest distance and closest velocity. Those thresholds are given in the settings file.

If a base match cannot be found for a tracking ID, the countdown will initiate. In the upcoming frames when the countdown reaches to (-1), the tracker ID will be deleted.

Also, if there is no best match tracking ID for a given boundary box or new tracking ID will be initialized for the box.

if (best\_match\_distance == float('inf')) or (force\_add) or (f == 0):

   vehicle\_ids, new\_id = self.generate\_traking\_id(vehicle\_ids, box, [0,0])

   #print(f"CREATE NEW BOX: {new\_id}")

   bb, bb\_res = self.create\_bb\_box(box, new\_id)

   o3d\_boxes.append(bb)

   box\_results.append(bb\_res)

### Continuous Tracking

If a best match tracking ID is found for a given boundary box, the tracking ID will be assigned for the boundary box followed by following actions.

* Update the Kalman filter and predict next location of the boundary box.
* Add the updated Kalman filter to the tracker (To be used in the next frame)

### Visualization in open3D

For troubleshooting purposes the predicted IDs will be visualized in open 3D. However this can be ignored by commenting the following line.

o3d.visualization.draw\_geometries(o3d\_boxes)

### Output

The output created in the same directory called “3d\_ann.json”. This has the same format as input file for CADC\_DevKit (<https://github.com/mpitropov/cadc_devkit>). However, there are additional parameters of tracking ID and color are added.

## How to generate BEV

File name: cadc\_devkit/run\_demo\_lidar\_dev2.py

BEV Video/GIF: Generates a birds-eye view representation of the object tracking results, allowing for visual inspection.

This is a modified version of “run\_demo\_lidar\_bev.py” by cadc\_toolkit.

**Output video can be created by:**

make\_video\_from\_images(image\_folder, output\_video\_file, fps)

**Output GIF can be created by:**

make\_gif\_from\_images(image\_folder, output\_gif\_file, duration)