FP.2 Compute Lidar-based TTC

Compute the time-to-collision (TTC) in second for all matched 3D objects **using only Lidar measurements** from the matched bounding boxes between current and previous frame. It is implemented in `computeTTCLidar` function. It involves the following steps:

1. **Statistical Robustness:** To achieve this, a clustering algorithm (similar to DBSCAN) is applied to the Lidar points using `pcl` library for both current and previous frames:
    a. First, Lidar points are converted to PointCloud data structures from `pcl` library.
    b. PointCloud data get clustered using `pcl::EuclideanClusterExtraction` with following parameters:

```
// Create a KD-Tree
pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ>);

// Create the clustering object (similar to DBSCAN algorithm)
pcl::EuclideanClusterExtraction<pcl::PointXYZ> ec;
ec.setClusterTolerance(0.15); // 20cm
ec.setMinClusterSize(50);
ec.setMaxClusterSize(2000);
ec.setSearchMethod(tree);
```

    function also reports how many points are discarded.
    c. Next, we loop over all the clusters found:
        a. Instead of finding the point with minimum distance, certain percentile is considered to reduce/prevent the influence of outliers.
            a. the percentile value is adapted based on how many points are in the given cluster:

```
int P;
if (N >= 100) {
    P = N/100; // 1st percentile
} else if (N >= 10) {
    P = N/10; // 10th percentile
} else {
    P = N/2; // median
}
```

2. **Computation of TTC:**
    a. `relVel` is calculated based on the points with minimum distance from the previous and current frames:

```
double dt = 1.0 / (frameRate + 1e-8);
double relVelX = (minDistanceXCurr - minDistanceXPrev) / (dt + 1e-8);
```

    b. If `relVel` is close to zero, we set `TTC = NAN` (catching the case to avoid division by zero)
    c. If `relVel` is not zero: `TTC = minDistanceXCurr / relVelX`

Summary of Results:

To compare the influence of the outliers, we plot the TTC results for both the naive computation with the robust computation:

> Note: we can further smooth the calculated `relVel` using **Polyak Averaging** to smooth the effect of outliers.

$$relVel_{smoothed} = 0.8 * relVel_{curr} \ + \ 0.2 * relVel_{smoothed-prev}$$

$$TTC_{Smoothed} = \frac{-X_{curr}}{relVel_{Smoothed}}$$

Table below is populated by hand using 3D object visualization function; `topview` of the Lidar points (to compute Naive-TTC).

Image ID	Naive-relVel	Naive-TTC	Robust-relVel	Robust-TTC	Robust-Smoothed-relVel	Robust-Smoothed-TTC
0						
1	-0.61	12.97	-0.57	13.89	-0.57	13.89
2	-0.64	12.26	-0.68	11.54	-0.66	11.9
3	-0.56	13.92	-0.44	17.74	-0.48	16.26
4	-1.08	7.12	-0.61	12.7	-0.58	13.36
5	-0.47	16.25	-0.62	12.39	-0.61	12.6
6	-0.61	12.42	-0.53	14.4	-0.55	13.87
7	-0.22	34.34	-0.7	10.8	-0.67	11.28
8	-0.61	12.42	-0.42	17.9	-0.47	16
9	-0.41	18.13	-0.48	15.56	-0.48	15.56
10	-0.41	18.03	-0.56	13.24	-0.54	13.73
11	-1.88	3.83	-0.65	11.31	-0.63	11.67
12	0.67	-10.85	-0.75	9.7	-0.73	9.97
13	-0.78	9.22	-0.79	9.11	-0.78	9.22
14	-0.65	10.97	-0.61	11.7	-0.64	11.15
15	-0.87	8.09	-0.9	7.83	-0.85	8.3
16	-2.15	3.18	-0.77	9.05	-0.79	8.82
17	0.69	-9.99	-0.71	9.71	-0.73	9.45
18	-0.82	8.31	-0.81	8.41	-0.79	8.63