Model-free Object Detection and Tracking

Roshan Benefo, Karel Smejkal

May 20, 2021

Abstract

The main goal of this project is to implement solution for model-free detection and tracking of moving objects using 2D laser scanner on the F1Tenth 1/10 scale autonomous race car system. Specifically, object tracking is achieved using 2D LiDAR of the F1Tenth vehicle. This project builds on the work of Dominic Zeng Wang and his collaborators, who proposed a model-free LiDAR-based object tracking system that separates out the static background to help track dynamic objects in a principled and straightforward way. We discuss the implementation approach and decisions that were used in development of the car and present an experimental evaluation of our solution. Given the restrictions posed by the pandemic, this project will heavily leverage virtual environments of the F1Tenth team's proprietary gym environment.

1 Introduction

The detection and tracking of moving objects is an important task in robotics. In particular, being able to robustly track other objects is critical in autonomous racing, allowing vehicles to accurately plan maneuvers around other cars and avoid obstacles. We have implemented an algorithm able to detect and then consistently track dynamic objects without any assumptions about the shape of these objects as well as recognize static points around the ego vehicle using 2D LiDAR. This model-free approach allows our algorithm to be used outside of just autonomous racing; it can be used in collision warning systems, people classification, or as an input to a generalized dynamic planner. We have aimed to make the system run real-time in order to be useful, especially if we want to apply it in the F1Tenth racing competitions.

The algorithmic steps of the implemented system include processing of incoming LiDAR and Odometry data, described in section 5.1 and its implementation more closely in 6.4 and 6.3. This is followed by prediction update step, again described in 5.1 and 5.2, composed of clustering, detailed in 6.4.2, coarse association, delineated in sections 5.4 6.4.3, fine association is outlined in sections 5.5 and 6.4.4, and an extended Kalman Filter update. The whole algorithm is finalized by transformation estimation and merging of existing associated objects or initialization of new objects described in 5.6. In addition we include the evaluation section 7.1, where we evaluate the run-time of algorithm and duration and precision of tracking in comparison to ground truth speed. Finally we will discuss the next steps and long term plan of improving the current implementation in section 8.

2 Related Work

Research in 2D LiDAR object detection and tracking is relatively limited in recent years and most of the recent papers are focused more on 3D LiDAR systems, as autonomous cars are adapting 3D sensors rather than 2D because of the increasing mobile computation power and demand for very accurate results of the algorithms running on these cars. However, 3D LiDAR systems remain too heavy and too bulky to be used on lightweight racecars like the F1Tenth racing system. To keep the list of related work narrow, we mention only classical (non-deep learning based approaches), which we find have us longer runtimes and are thus not appropriate for autonomous racing as classical approaches.

A few authors have proposed algorithms for dealing with object detection and tracking for 2D LiDAR. In some of the older work, Mendes et al. (2004) [1] use their multitarget detection and tracking system for collision avoidance. Zhao et al. (2006) [2] is outdoor application of monitoring an intersection. They

propose a joint tracking and classification algorithm for moving objects. Their method of joint tracking and classification developed by processing on laser scan data, where moving objects are classified depending on 3 classes of objects based on how many axis they have. An object model is defined by picking up the appearances (Markov states) of an object in an instant laser measurement, and modeling each traffic flow as the transition along a chain of the Markov states (Markov chain). These proposals, however, are not suitable for the task of object tracking in autonomous racing, where vehicles need to be able to identify discrete dynamic objects and separate them out from the static background.

Other methods like detecting non-static objects by occupation of previously unoccupied space are proposed in Lindstrom and Eklundh (2001) [3]. There is a lot of similar work which is based on an occupancy grid method and joint probabilistic data association filters. Petrovaskaya (2011) [4] proposes a model-based framework for detecting vehicles which assumes that they have a rectangular shape, and relies on a 'virtual scan', a representation. Lastly, our worked is based on paper from Wang et al. (2015) [5], which uses all the algorithms described in this paper to achieve model-free object tracking and most of our ideas are following on this work and trying to improve it.

3 Problem Statement

3.1 Overall Formulation

We aim to design a system for the F1Tenth vehicle that can use its 2D LiDAR to identify objects, track them, and perform coarse classification (determining whether or not they are static or dynamic) on these objects. In particular, we aim to do so in a real-time manner, with a system update rate over approximately 10 Hz, approximately four times the update rate of the Hoyuko LiDAR onboard the vehicle.

Our task is purely perception; as such, we assume that the ego vehicle has at all times an accurate knowledge of its state - that is, its state estimation is corrupted by only low magnitude noise.

3.2 Vehicle Equipment

The F1Tenth vehicle LiDAR is the Hoyuko UTM-30LX LiDAR, which provides a 270° scanning range, and can see up to 30 meters. Its scanning frequency is 40Hz, and has an accuracy of $\pm 30mm$ up to 10 meters, and $\pm 50mm$ from 10 meters to 30 meters. The sensor datasheet is available here.

4 Contributions

We introduce a novel framework for transforming points associated together with data association into fictious measurements used to update the state of tracked objects. Furthermore, we propose a novel coarse association estimation algorithm that is faster and more robust than the ICP algorithm proposed by [5].

Third, we integrate a free-space removal algorithm into the framework proposed by [5]. This helps generate more robust state estimations by taking advantage of the fact that LiDAR readings tell us two things– first, they give information that can help adjust the likelihood of an object being at a location that they impinge, and second, they help reduce the likelihood of objects being *between* the LiDAR impingement and the ego vehicle.

Lastly, we make several speed optimizations to the system to allow it to run in real time on an F1Tenth racecar at around 10 FPS. These include speed-increasing approximations made within the JCBB-Refine Fine Association algorithm, and the use of the Ramer - Douglas - Peucker algorithm to reduce the number of tracked boundary points, thus increasing the speed of algorithms (like JCBB) whose runtimes are functions of the number of incoming points.

5 System overview

Our system comprises of three main parts: data association, state update, and merging. Data association involves a clustering algorithm which separates incoming LiDAR point clouds into separate components, and two association algorithms. State update includes a transformation estimation algorithm to calculate a

state update measurement from associated points, and an Extended Kalman Filter to provide a framework for how measurements are used to update state. Finally, merging allows separately tracked objects to be combined if they are part of the same object. We go into all of these in the following sections.

5.1 Sensor pose prediction on odometry measurement

Individual laser scan beams and a set of odometry measurements are used as input to the algorithm. Without odometry information, the system would not be able to distinguish static objects as all motion estimates are relative to the LiDAR sensor. Odometry information provides us with an estimate of speed v of non-holonomic vehicle, mean angle θ of front wheels. We get to the prediction model of the sensor as follows.

Considering that the speed of vehicle v and the θ are constant during prediction time Δt, the uncertain control input is defined as follow

$$\mathbf{u}(v,\theta) = \begin{bmatrix} \Delta l \\ \Delta \Psi \end{bmatrix} = \begin{bmatrix} v\Delta t \\ \omega \Delta t \end{bmatrix} = v\Delta t \begin{bmatrix} 1 \\ \frac{\tan\theta}{L} \end{bmatrix} \tag{1}$$

where L being the distance between front and rear wheel axles, ω is the angular speed of the vehicle. We also assume that the distance L is constant. To get Jacobian of the control input, we differentiate the equation (1)

$$\mathbf{U}(v,\theta) = \Delta t \begin{bmatrix} 1 & 0 \\ \frac{\tan\theta}{L} & \frac{vsec^2\theta}{L} \end{bmatrix} \tag{2}$$

We assume that the vehicle state is corrupted by noise covariance matrix $\mathbf{V}$,

$$\mathbf{V} = \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\theta^2 \end{bmatrix} \tag{3}$$

Then, the vehicle state has mean $\widehat{\mathbf{u}} = \mathbf{u}(\hat{v}, \hat{\theta})$ and covariance $\mathbf{Q} = \mathbf{UVU}^T$[5].

5.2 Dynamic object motion prediction

When new measurements are received, currently existing dynamic tracks are predicted forward using generic motion prediction model before they are updated with the newly received scans. We assume a constant velocity model, because we do not know any information about the object and thus it's motion pattern. This is disadvantage of working with model-free assumption. We also model not only constant linear velocity component, but angular velocity component as well. This should make motion prediction model able to capture a full range of 2D rigid body motions [5].

5.3 Hierarchical data association

It is not possible to directly observe all state variables in the joint state and for those which we can directly observe, more specifically boundary points belonging to static or dynamic object, it is difficult to tell which one do they belong to or if it is a new boundary point, which haven't been seen before. Each time we get new set of LiDAR scan, we have to determine following:

1. If new measurement is a part static object:
   - Is it already existing boundary point?
   - Is it a new boundary point?
2. If new measurement is a part of existing dynamic object:
   - Is it already existing boundary point on the object? If that's the case it also needs to be determined which specific object this measurement belongs to.
   - Is it a new boundary point on the object?
3. Is new measurement part of new, yet unassociated object?

In order to solve data association problem we have to associate the measurements on low and high level. Meaning, on high level we want to firstly create rough associations on coarse scale, where we divide all newly observed measurements into clusters and find out for each cluster if it belongs to already existing static background or dynamic object. If neither is the case than we assign it as a new, yet unobserved object. On low level we create fine associations, where the each measurements within each cluster is either associated to yet exiting boundary point or used to initialized a new point.

5.4 Coarse-level data association

In the previous section we discussed that in order to decide if the new set of measurements belongs to boundaries of already existing objects or if the measurements are new objects, we need to break down the problem into coarse and fine level association. We mentioned as well that in coarse association we want to divide incoming measurements into clusters. More specifically, we divide incoming laser scan data into set of clusters $C = \{C_1, C_2, ..., C_{|C|}\}$.

Once we have the data separated into clusters, we run our own modified version of Iterative Closest Point Algorithm (ICP) [6]. [5] recommends to estimate association validity based on an ICP procedure that involves a nearest neighbor estimation. This simple nearest neighbor procedure means that for many shapes, all points in one object are associated with a single point in another. This yields bad transformation estimation. This can be seen on the Figure 1.

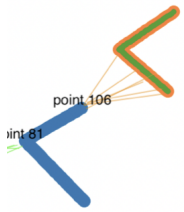


Figure 1: The problem using nearest neighbour in ICP, as it can generate 1-n association of points, which leads to bad transformation estimation between objects. In this case point 106 in the blue object is assigned as the closest point to every point in the green object, this leads to wrong calculation of transformation between these 2 objects.

Rather, we propose a new estimation based on minimizing the total distance between a group of pairs, where we make sure that if we assign point from one object to point in another object, these points are not considered again as candidates for another point pair. Using this approach, we have seen significant improvement compared to the [5] not only in terms of precision, but speed of convergence of ICP as well.

With this version of ICP, we align the incoming lidar scan with clusters which already has been associated to static background. After that, we calculate a transformation between the corresponding points: if the transformation (either its translation magnitude or rotation magnitude) is too large, then the association is rejected and the cluster deemed to not be a good match with the candidate object. Associated clusters that are assigned static background are removed from the set of clusters C. Then, we repeat the same procedure on the remaining clusters in C, however this time we are trying to match incoming clusters with already known dynamic objects using ICP. The paper [5] recommends a procedure where we remove clusters from our consideration set after they have been associated with a dynamic track, as we step through the dynamic tracks in order. This allows early tracks to "gobble up" clusters. Rather, we just mark already assigned dynamic clusters as already used clusters variable. If there are still some clusters remaining in C, which are not in the used clusters variable, we mark them as objects we have not seen yet and later intialize them as new tracks.

ICP is known to be slow when the initial misalignment between two objects is large, therefore we use a parameter to look only within certain distance of the cluster we are trying to match.

To get better intuition about how coarse association works, the whole procedure is shown in Algorithm 1.

Algorithm 1: Coarse Association

$C \leftarrow Clustering(Z)$;
for *i in range(# of C)* **do**
  $(\widehat{x}, P, A_s) \leftarrow AssociateAndUpdateWithStatic(\widehat{x}, P, C, i)$;
end
$C \leftarrow C \setminus A_s$;
for *i in range(# of C)* **do**
  $(\widehat{x}, P, A_d) \leftarrow AssociateAndUpdateWithDynamic(\widehat{x}, P, C, i)$
end
$Already\_Used \leftarrow A_d$;
for *i in range(# of C)* **do**
  if C_i *not in Already_Used* **then**
    $(\widehat{x}, P, A_n) \leftarrow InitializeNewTrack(\widehat{x}, P, C, i)$
  end
end
return A_s, A_d, A_n

5.5 Fine-level data association

While coarse-level data association helps us identify which *clusters* of LiDAR points are associated with which objects (and which ones are possibly of new objects), in order to update the state of our currently tracked objects, we need to perform point-to-point matching. Successfully performing this matching is critical to the entire system, since we use these matches to estimate a transformation that is used to update the state of our tracked objects.

In fine-level data association, we perform such matching. The core algorithm in this stage is the Joint Compatibility Branch and Bound (JCBB) algorithm, a common algorithm used for MOT (multi-object tracking) data association. In the JCBB algorithm, we run a depth first search to find a joint data association that minimizes jNIS (the joint normalized innovation squared).

The formula for jNIS is as follows. Say we have a set of measurements $\hat{z}_\sigma$ that we propose are associated with a set of currently estimated boundary points $\hat{h}_\sigma$. The joint pairing has an innovation covariance of:

$$S_\sigma = \begin{bmatrix} H_{\sigma(1)} P H_{\sigma(1)}^T + R & H_{\sigma(1)} P H_{\sigma(2)}^T & \cdots \\ H_{\sigma(2)} P H_{\sigma(1)}^T & H_{\sigma(2)} P H_{\sigma(2)}^T + R & \\ \vdots & \vdots & \ddots \end{bmatrix}$$

Where P is the covariance of the track, R is the expected measurement noise of the LiDAR, and H_σ is the Jacobian of the measurement model of the boundary points. Then, the jNIS is:

$$jNIS = (\hat{z}_\sigma - h_\sigma)^T S_\sigma^{-1} (\hat{z}_\sigma - h_\sigma)$$

This formula is similar to the Mahalanobis distance, which measures the distance between two probability distributions.

5.6 Transformation Estimation

Fine association outputs a set of paired scan points and currently estimated boundary points. These pairs are used to update the hidden state of our tracks.

To do this, we diverge from the paper, and use these pair points to estimate a transformation **T** between the estimated boundary points and scan points. Given that we know the timestep between observations, we can use **T** to create a fictious measurement of the track's position, orientation, and velocity.

A rigid body transformation is appropriate in this use-case, as we are designing this system for F1Tenth racing, where the only tracks we will likely encounter are other cars, walls, or cones, all relatively rigid

objects. This assumption, however, may not be appropriate if this system were deployed on a vehicle that may observe non-rigid objects like humans or animals.

As we are estimating a 2D rigid body transformation, **T** will simply be a 3x3 matrix that encodes the SE(2) transformation that best maps our scan points to our estimated boundary points. To best estimate this transformation, we can either estimate a least squares solution to calculate the transformation matrix between the entire set of pairs via the Kabsch algorithm, or as a cheaper alternative which performs better when the number of pairs is high, run RANSAC, and compute the transformation matrix for a minibatch of pairs (via Kabsch), keeping the transformation that generates the highest number of inliers on the entire set of pairs.

Algorithm 2: RANSAC Estimation of Rigid Transformation

$bestNumInliers \leftarrow 0$;
$T_{best} \leftarrow I(3)$;
for *i in range(iterations)* **do**
  $scans_m \leftarrow minibatch(scans)$;
  $boundaries_m \leftarrow minibatch(boundaries)$;
  $T_{candidate} \leftarrow calcTransformation(boundaries_m, scans_m)$;
  $numInliers \leftarrow calcInliers(boundaries, scans, T_{candidate})$;
  if $numInliers > bestNumInliers$ **then**
    $bestNumInliers \leftarrow numInliers$;
    $T_{best} \leftarrow T_{candidate}$;
  end
end
return T_{best}

Once the transformation matrix is estimated, it can be used to approximate the fictious measurement by assuming that the transformation occurred as the track was moving with a constant linear and angular velocity.

Algorithm 3: Measurement Calculation from Transformation Estimation

$translation \leftarrow getTranslation(T_{best})$;
$rotation \leftarrow getRotation(T_{best})$;
$measurement.position \leftarrow track.position + translation$;
$si \leftarrow sign(track.orientation + rotation)$;
$measurement.orientation \leftarrow si * (\text{track.orientation+rotation}) mod \pi$;
$measurement.velocity \leftarrow translation/dt + track.velocity$;
$measurement.angularVelocity \leftarrow rotation/dt + track.angularVelocity$;
return measurement

It is important to note that we only perform transformation estimation if the number of point pairs if greater than 4, as accurate transformation estimation is difficult with only one a few sets of paired points (rotation and translation estimation diverges in these cases).

5.7 Measurement Model

Our 'fictious' measurements can be thought of direct observations of the position, velocity, and orientation of tracked objects. These measurements, however, are quite noisy, as data association and transformation estimation both are inherently noisy processes. As such, our measurement model can be thought of as following the below equation, where R is a diagonal noise covariance matrix. In particular, the terms in R associated with velocity (angular and linear) have noise terms that are *larger* than those associated with position, as velocity estimation is likely to be more noisy that position estimation.

$$y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} x + R$$

A more robust treatment of the problem may involve looking at the correlation between velocity and

position measurements, thus making placing off-diagonal elements in R. Experimentally, however, we find that treating R as a diagonal matrix generates sufficiently accurate tracking.

5.8 Track merging

During each LiDAR update iteration, we check to determine whether or not any currently tracked object is either part of the static background, or part of another dynamic object. In this, we diverge from original paper and check **all** objects, rather than just the ones newly initialized. We find experimentally that this allows the algorithm to correct previous mistakes by merging tracks that were previously, and mistakenly, not merged.

To perform the merging step, we calculate the Mahalanobis distance between each track's state and the merging target's state, taking into account the track's covariance. We first test to see whether or not each track should be merged with the static background by comparing the track's velocity with a proposed value of 0 (static objects have a velocity of 0). To determine whether or not a track is part of the static background, we compute the Mahalanobis distance between the track's velocity probability distribution and 0, and compare the distance against a chi squared significance value. If the distance is lower than the chi squared value, then we merge the track with the static background.

The remaining tracks are then tested against other dynamic tracks to see whether or not they should be merged. The merging procedure works here nearly the same as the procedure for the static background: rather than comparing the track's velocity, however, we compare the track's position and velocity against the proposed merge target's. If the calculated Mahalanobis distance between these two distributions falls below the chi squared significance value, then these tracks are merged.

6 Implementation

6.1 State Structure

In order to store information about our tracked dynamic objects, newly seen objects and static background we need to create a system representation of their state. For this we created separate class State and Track to allow us keep the information about the state of specific object, independently of its shape. We cannot observe the motions of dynamic objects directly as the sensor is constantly moving, however we can measure the correlations between different objects, if the states of the objects have been estimated in a single joint distribution. We can also simultaneously estimate local static background as part of the joint state. The state therefore consists of three parts: the ego vehicle pose, the dynamic objects and the static background.

In the code we represent the sensor pose estimation (the ego vehicle pose) as xs and its covariance Pxc. We also need transform between the sensor and vehicle's frames of reference as holonomic constraints apply to the vehicle but not to the sensor and odometry measurements are referenced to the vehicle's frame of reference. This transformation is saved in variable xc and its covariance in Pxc. On top of that we create two classes StaticTrack and DynamicTrack with Track class as their parent, to differentiate between static background and dynamic objects in easy manner. Class Track keeps track of how many times have we seen the specific object based on its position in world frame of reference and if it reaches it maturity threshold (being observed for fixed number of frames in a row), we test it against the static background and each currently known dynamic track and they merged accordingly into their assigned class. Each object has its own Track class, which also stores its boundary points, in the object reference frame. The object reference frame is chosen to be estimated the centroid of the object when the algorithm first detected it it. That is, even if we see more of the object later, we maintain its centroid for consistency.

6.2 Reference Frames

All reference frames are orientations are fixed to the world frame orientation for consistency. However, as mentioned previously, each object has a local reference frame that is centered on its own centroid.

6.3 Odometry Update

As we assume that the vehicle has a good estimation of its state at all time, the Odometry update is relatively trivialâ in simulation, we simply take the true pose of the vehicle, and add some Gaussian noise to it to represent small amounts of noise in the vehicle state estimation.

6.4 LiDAR Update

Our LiDAR update consists of a number of steps enumerated in Figure 2. It runs whenever the previous LiDAR computation is completed, or when we receive a new LiDAR reading, whichever is sooner. After forward propagation of all states in every stored Kalman Filter, we then run our clean up states algorithm, which simplifies the number of stored points for faster processing, removes points that are out of view of the LiDAR sensor, and removes points that are within free spaces identified by the current LiDAR scan. Then, the incoming LiDAR data is clustered and sent into the coarse association algorithm, where clusters are matched with currently tracked static or dynamic objects. Unassociated clusters are treated as new tracks and are initialized; associated clusters are sent into the fine association algorithm where point-to-point correspondences are calculated. After this, a transformation estimate is calculated relating the paired scan points to the previously tracked static and dynamic objects, and the corresponding objects are updatedâ either with an EKF update if the object is a dynamic object, or with a simple concatenation, if the object is a static object. Finally, tracked dynamic objects are evaluated to check if they should be merged into the static background, or if they should be merged with each other.

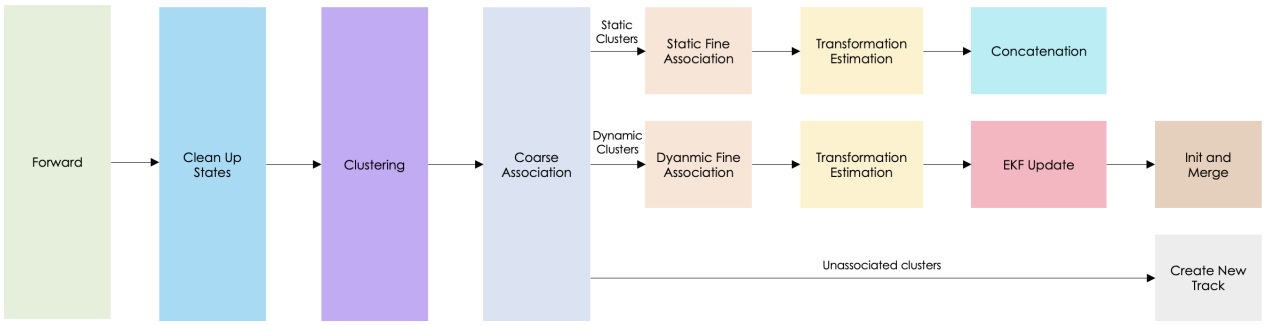


Figure 2: Diagram of LiDAR Update sequence.

6.4.1 Clean up states

In order to ease memory requirements and speed up the algorithm, we need to clean up the points and objects which are no longer relevant to us. This is done by CleanUpStates class, where we remove all points which drop out of field of view of LiDAR or out outdated, meaning they have not been seen for a while.

Remove old and free If some of clusters (objects) have not been seen for a fixed amount of frames in a row, we mark them as outdated and remove them from our system. This again helps speed up the algorithm as well as ensures that we are not associating incoming LiDAR points with out–of –date objects. Furthermore, we take advantage of the fact that LiDAR impingements give us two pieces of informationâ first, that an object is likely to be where the LiDAR hit, and that **there are no objects between the vehicle and that object**. As such, during CleanUpStates, we remove objects that are between the vehicle and current LiDAR impingements.

Remove out of range or out of field of view The Hoyuko UTM-30LX LiDAR offers 270° scanning range so any point and object between 135° - 180° to left and between 135° - 180° to right of the heading of the vehicle is removed. This assures that we still keep track of everything which is relevant and in front of the ego vehicle while not accumulating too many points and objects which are out of date.

Ramer–Douglas–Peucker algorithm Very simple but efficient and easy to implement improvement to the original paper [5] is inclusion of line simplification, specifically Ramer–Douglas–Peucker algorithm (RDP). This idea came from the effort to speed up fine association algorithm. RDP simply remove excess points

while keeping the shape of the object. This is illustrated on following figure 3. We decided to use an RDP line simplification library for Python that uses a Rust binary to achieve the best run-time possible.

The algorithm works by recursively dividing the line starting with all points in–between initial and end point. After that it is trying to find the farthest point from the line segment using the initial and last points as end points. If the point is closer than ϵ (distance dimension) to the line segment, then any points not currently marked to be kept can be discarded without the simplified curve being worse than ϵ. If the point farthest from the line segment is greater than ϵ from the approximation then that point must be kept. The algorithm recursively calls itself with the first point and the farthest point and then with the farthest point and the last point, which includes the farthest point being marked as kept. When the recursion is completed a new output curve can be generated consisting of all and only those points that have been marked as kept [7]. The higher the parameter ϵ the more points will be deleted resulting in less accurate shape of the original object. This can lead to problems with estimating correct transformation in the fine association as the two shapes can be quite different depending on the how aggressively RDP was. However after experimenting with different values for ϵ we found a fixed value which is aggressive enough to lead to significant speedup of the algorithm, while not affecting correctness of fine association.

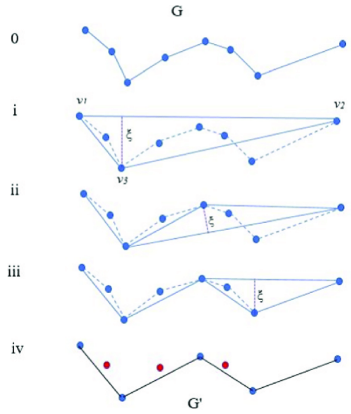


Figure 3: Ramer–Douglas–Peucker algorithm visualization [8]. In the step 0 we have a shape consisting of 8 points which have been reduced to only 5 points in step 4.

6.4.2 Clustering

Our clustering algorithm comes in two parts: first, we construct a euclidean minimum spanning tree (EMST) of the laser scan points, and then search through that tree to generate clusters via the Efficient Graph Based Segmentation Algorithm (EGBIS) [9].

To create the EMST, we first generated a Delauney triangulation of the point set (an $O(nlogn)$ operation) to form a graph, and labeled each edge with its length. Then, we used Kruskal's algorithm to search through that graph and form a minimum spanning tree, using a disjoint set data structure to improve runtime. While our initial implementation of this algorithm was in Python, we eventually switched over to an EMST implementation written in C++ from mlpack to improve runtime.

Once the minimum spanning tree is created, we then search through that tree to generate a coherent set of clusters via the EGBIS algorithm. In this, we created a Python implementation of the C++ EGBIS implementation written by the original authors, and stored the output clusters as a dictionary, with each cluster assigned a unique key mapping to a set of stored laser scan indexes.

The EGBIS algorithm has a parameter k that determines the threshold required for a cluster to be unique. The lower the k, the more clusters are generated– the higher the k, the fewer. In practice, by tuning k, we are able to consistently generate robust sets of clusters.

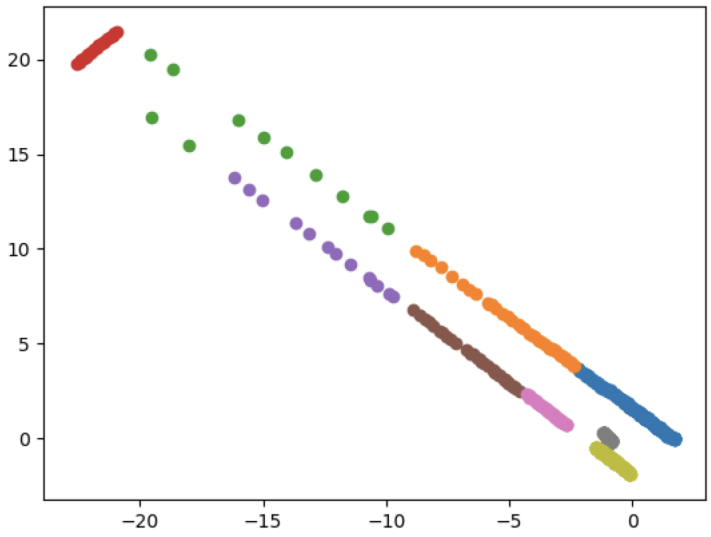
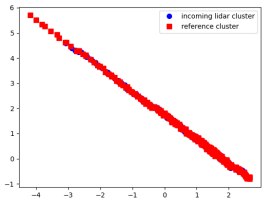


Figure 4: Example clustering from our EMST-EGBIS clustering algorithm. The tracked vehicle is the grey blob in the bottom right.

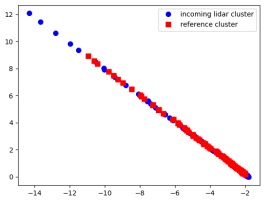
6.4.3 Coarse Association and ICP

As already mentioned, the purpose of coarse association is for algorithm to get and idea which objects in the current LiDAR scan belongs to already existing objects and static background. It is based on modified version Iterative Closest Point Algorithm (ICP) [6], which we are going to describe more closely in this section.

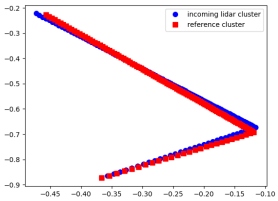
ICP is efficient algorithm for minimizing the difference between two clusters of points. In robotics it is mainly used for reconstructing 2D or 3D surfaces from different scans, to localize robots and achieve optimal path planning [10]. In our work we use it to align two clusters within certain distance threshold and compute rough guess of the rigid transformation between target and source clusters. If both translation and rotation between these two clusters are lower than very small, fixed threshold within multiple iteration, we say the the target cluster is corresponding to the source cluster. ICP can be very slow if we have big initial misalignment between target and source, however with scanning frequency of out LiDAR 40Hz we are making sure that the distance and misalignment between two objects can't be so high that ICP would fail to converge quickly, given our prediction model is correct as the ICP is run right after the prediction step. The changes we have made to original ICP and the pseudo-algorithm of coarse association are described in section 5.4. The resulting transformation between found point pairs from target and source cluster is done by scikit-image library using Euclidean Transformation. The result of our associations computed by ICP on clusters from figure 4 is shown figures 5.



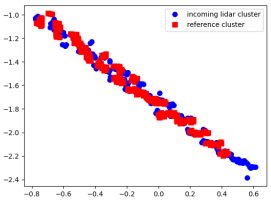
(a) Blue cluster



(b) Orange cluster



(c) Gray cluster (target vehicle)



(d) Yellow cluster

Figure 5: Found associations in between clusters from LiDAR scan (shown on fig 4) and already known objects from previous frame

6.4.4 JCBB

The JCBB fine association algorithm is comprised of three parts. First, we construct an initial association based on the output of coarse association– scan points and estimated boundary points that are sent into an array to form our initial association.

Individual Association: For every scan point, and every estimated boundary point, we then compute their potential individual compatibility. This allows the later Depth First Search to operate efficiently, by pre-rejecting incompatible points (typically ones that are far away from each other). The initial association is then pruned based on these individual compatibilities.

Prune Association: We also wish that our output association has a one-to-one pairing between boundary points and scan points. That is, each scan point can only be paired to at once one boundary point, and each boundary point can only be paired to at once one scan point. Thus, the initial association is further refined to remove double-pairing – in cases where two scan points are paired to one boundary point (or vice versa), we simply clear the association for both points.

Minimal Association: Then, we run through the association, and iteratively remove pairs from the association to yield the largest jNIS decrease. We continue doing this until the total jNIS of the association falls below the chi squared significance value. In practice, this operation is computationally expensive (it requires lots of iterations), so we cut this portion of the algorithm off if the number of iterations exceeds a max iteration parameter.

Depth First Search: With our minimal association, we begin adding points to the association to generate a valid association (one whose jNIS falls below the chi squared significance value) with the maximum number of pairs. If there are multiple associations with the same number of pairs, we choose the one with the lowest jNIS value.

In practice, we implement this portion of the algorithm as a depth first search. A diagram of a sample search tree is shown below. As can be seen, each level of the tree refers to a scan point – as we go through the tree, we search through sets of boundary points that are associated with our scan points. Every level of the tree has an option for its scan point to be associated with no boundary points, in which case it becomes associated with a "nAn" (the red circles in the below plot). At every point in the tree, we recalculate the jNIS of the proposed association thus far.

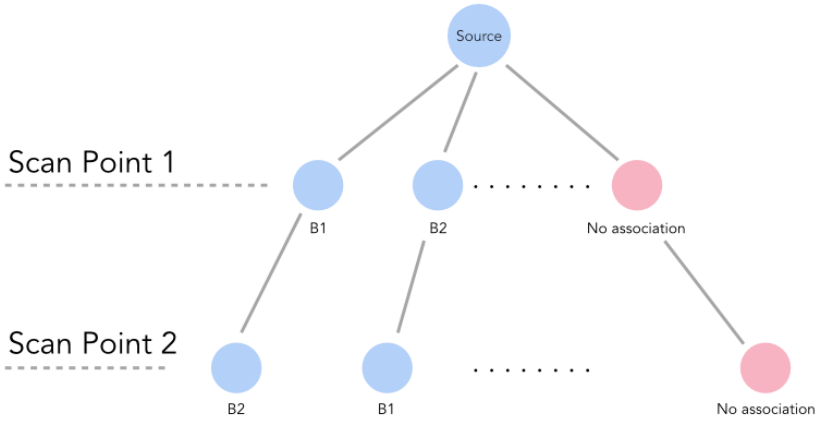
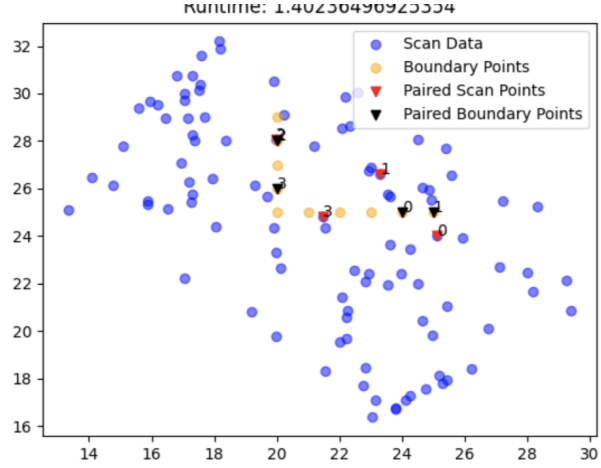


Figure 6: Tree structure for JCBB DFS. Nodes labeled as "B" represent boundary points.

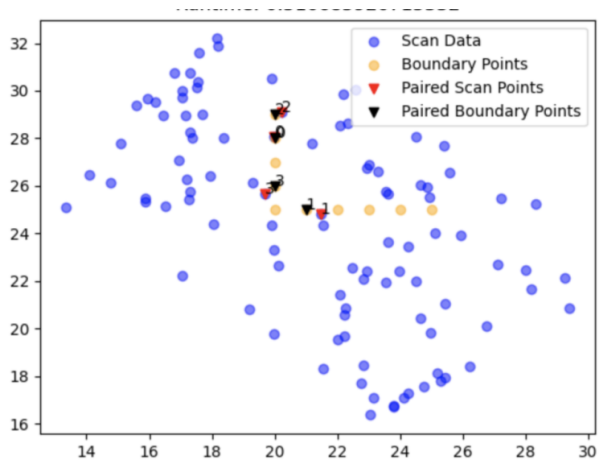
To improve the efficiency of the DFS search, we prune portions of the tree. jNIS is a monotonically increasing function – as we increase the number of pairs in our association, the jNIS will always increase. As such, if, while going down a branch of the tree, we see that our jNIS is higher than the chi squared significance value, we cut off the remainder of the branch.

We further prune our search tree by maintaining the best jNIS and highest number of associations as class variables. As such, once again, if we see that our jNIS is higher than the best recorded jNIS, we cut off the search of that particular branch.

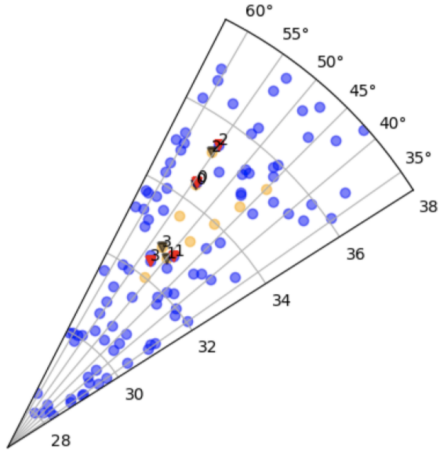
Polar to Cartesian: The original paper [5] recommends calculating jNIS based on polar coordinates; as such, the distance between points is measured as the distance between their r and θ values in a polar graph. However, we find that calculating jNIS in Cartesian coordinates yields far better associations, due to the fact that when the measured points are far away from the ego vehicle, differences that seem small in Polar space are actually large in Cartesian space. This is shown in Figure 7c.



(a) Matches when using polar version of JCBB.



(b) Matches when using cartesian version of JCBB.



(c) Polar matches in polar coordinates.

Figure 7: Matches using polar version of JCBB vs cartesian. As can be seen, the Cartesian JCBB generates far better matches than the polar version. This is due to the fact that while the polar matches are close to each other in polar space (as seen in Image C), all of the scan points are far away from the ego vehicle, and as such, small changes in angle θ yield large changes in the Cartesian world coordinates of the matched points.

Speed Improvements: The JCBB is the slowest part of the system by far. As such, we experimented with a number of methods to increase its speed.

In practice, the DFS is the most time-consuming part of JCBB, as the search space of the DFS (even with pruning) can increasing dramatically if the number of scan points or boundary points is high. As such, we limit the number of recursions through the DFS– if we hit this limit, we simply return the best association generated thus far.

One of the most time-consuming parts of the DFS is the repeated recalculation of the jNIS. This recalculation involves the inversion of the covariance matrix S_σ, which can get computationally expensive if S_σ is large.

To help speed up this calculation, we take advantage of the fact that the jNIS can be calculated in triangular form. Instead of inverting S_σ each time we run through the DFS, we instead invert a covariance matrix S with **all** the boundary points and all the scan points at the very beginning of the algorithm (during the individual compatibility step). Then, we take its cholesky and store it as a class variable.

S_σ can be approximated by taking individual rows and columns from the stored cholesky matrix. We

can then solve for the jNIS via the following algorithm:

Algorithm 4: jNIS Rapid Approximation

$idxs \leftarrow pairedBoundaryPointIdxs$;
if *firstRun* **then**
  $S \leftarrow calcCovariance()$;
  $L \leftarrow cholesky(S)$;
  store(L);
else
  $L_\sigma \leftarrow L[idxs, idxs]$;
  $a \leftarrow (\hat{z}_\sigma - h_\sigma)$;
  $y \leftarrow solveTriangular(L_\sigma, a)$;
  $jNIS \leftarrow norm(y) * *2$
end
return jNIS

While the resulting jNIS is not completely accurate, as the inverse of portions of the matrix S cannot be completely accurately determined by taking rows and columns from its cholesky, we find that the speed increase (around 10x) yielded by using this approximation exceeds the resulting loss of accuracy.

To further improve the speed of JCBB, we wrote a version of it using Numba. We find that this sped up the implementation by around 1.5x âhowever, on certain frames, the algorithm can slow down quite a bit. Below is a plot of the loop run times taken from the two systems (the non-Numba JCBB, and the numba JCBB):

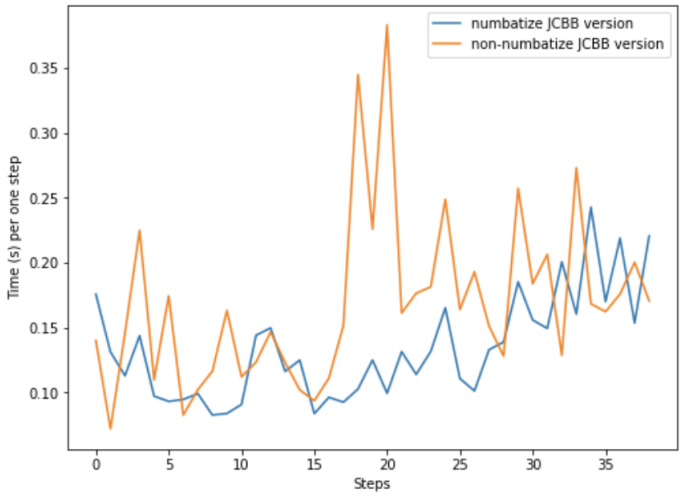


Figure 8: Algorithm Runtime comparison using JCBB Numba vs Non-Numba

Lastly, we attempted to remove the JCBB entirely from the algorithm, by replacing it with a simple algorithm that minimizes the joint distance between two sets of points, and rejects pairings that are too far apart. In practice, we find that this implementation does not generate good associations, causing rapid divergence in the algorithm output.

6.4.5 Static Update

Accurately performing data association for static objects is critical to avoiding system divergence – static objects that are incorrectly associated can "drift", appearing to the system to move, even if in reality they are not moving. This problem is particularly salient for smooth, large objects like walls, where a few incorrect associations can easily result in the entire wall appearing to shift.

To update the state of our static objects, we calculate a transformation between the paired scan points and their estimated boundary points. Before calculating this transformation, we center the two pairs of

points in a common reference frame centered at (0,0), to generate a more accurate transformation. Then, if the norm of computed translation is below a certain threshold (suggesting that the object has not moved), we simply concatenate the paired scan points with the currently estimated boundary points.

6.4.6 Dynamic Update

To update the state of our dynamic objects, we similarly calculate a transformation between our scan points and dynamic points, after centering both in a common reference frame centered at (0,0). We then feed this transformation into Algorithm 3, and send the resulting fictious measurement into our Kalman Filter. We only calculate a transformation and update the state of the dynamic object if the number of pairs is greater than 1, to ensure that we get an accurate transformation estimation.

6.4.7 Kalman Filter

While a robust implementation of this tracking system would call for the use of a single Kalman Filter (and single covariance and state matrix) for all tracked objects, in practice, we find that this approach is infeasible for implementation. Not only does it involve constantly adjusting the size of a large covariance matrix when tracks are added or dropped, but operations such as the matrix inverse begin to take a large amount of time as the matrix size increases. As such, we instead create an individual Kalman Filter for every object that holds their state and covariance.

When a LiDAR reading is received, the Kalman Filter performs its forward step. We calculate the dynamics matrix for forward propagation based on the time since the last LiDAR measurement. After transformation estimations are calculated for dynamic objects, they are sent into the Kalman Filter to perform an observation update. To speed up computation, we only run a filter on dynamic objects (we let static objects maintain a constant covariance, and assume that their state never changes as we believe them to not be moving). This, sometimes, may result in objects being incorrectly identified as static ââ however, experimentally, we find that CleanUpStates is able to quickly remove these objects when they are misidentified.

During experimentation, we played around with passing a custom noise covariance into the Kalman Filter, depending on the final jNIS of the output association from the JCBB, and the percentage of points that were associated. We found, however, that this did not improve our tracking accuracy by a significant amount.

6.4.8 Filter Smugness

The estimations provided by our Kalman Filter and Fine Association algorithm can easily diverge due to a problem called Filter Smugness [11], where the filter estimated covariance drops considerably such that it begins rejecting reasonable observations. We find that this oftentimes occurs when a tracked object accelerates from rest ââ while the object is not moving, the filter typically easily track it, and as more and more measurements come in that confirm the filter's previous state estimation, the filter's estimated covariance drops. However, if, by the time the object begins to accelerate, the covariance is sufficiently low, JCBB rejects scan points of the object, as the resulting jNIS (which is a function of the inverse of the track covariance) is too high. We find, experimentally, that this can be rectified via two methods. First, the dynamics noise in the filter dynamics propagation can be increased to prevent the covariance from dropping too low. Second, when an object is *not* associated with any scan points during one pass of the algorithm, we can artificially increase its estimated covariance such that it has a better chance of being associated with during the next pass.

6.4.9 Track Merging

Every iteration, we check all of the dynamic objects that have been seen more than 5 times whether or not it is a part of the static background, or a part of another dynamic object. We check only after an object has been observed for 5 iterations to allow the filter estimation of its state to converge.

To do this, we first check every object to see if it is part of the static background. Following the proposal by [5], we create a fictitious velocity measurement of $z = [0, 0, 0]$, where the elements of z correspond to the x

velocity, y velocity, and angular velocity. We then calculate the Mahalanobis distance between the estimated velocity of each object against this fictitious measurement. For an individual object, if this distance falls below a significance threshold, we merge this object with the static background.

Tracks that are not merged with the static object are similarly compared against each other to determine if they should be merged. This time, however, we compare **both** the velocities and positions of each trackâ if the Mahalanobis distance between the state estimations of two tracks fall below a confidence threshold, they are merged: the boundary points from one are concatenated with the boundary points from the other, and their centroids and states averaged.

In practice, we find that while the static merging works quite well (dynamic objects are rarely treated as part of the static background, unless they are moving very slowly, and objects that are part of the static background are usually quickly identified as static), dynamic merging does not. We discuss some of these issues in the Evaluation section.

7 Evaluation

In this section we evaluate our algorithm qualitatively, and then quantitatively in terms of speed, duration (how long we can track a dynamic object), and precision of tracking of one target vehicle.

7.1 Qualitative Evaluation

Figure 9 contains a sample output of the algorithm. A longer output video can be found here (note: the algorithm rate is significantly slower when plotting is enabled, as plotting is running on the same thread as the algorithm itself, an issue we bring up in the Conclusions section). In this output, we see that we are able to easily distinguish between static and dynamic objects (the static background is colored black), and are also able to consistently track dynamic objects for many frames. However, problematically, we do occasionally lose the dynamic object, and are unable to merge the newly created track representation that replaces the lost object with the previously lost object. The latter is because of filter smugness for high speed objects; we find that we consistently underestimate the velocity covariance for high speed objects. As such, when we calculate the Mahalanobis distance between two state estimates of two different objects for merging in our dynamic merging code, we typically reject the merger as the calculated distance is too high.

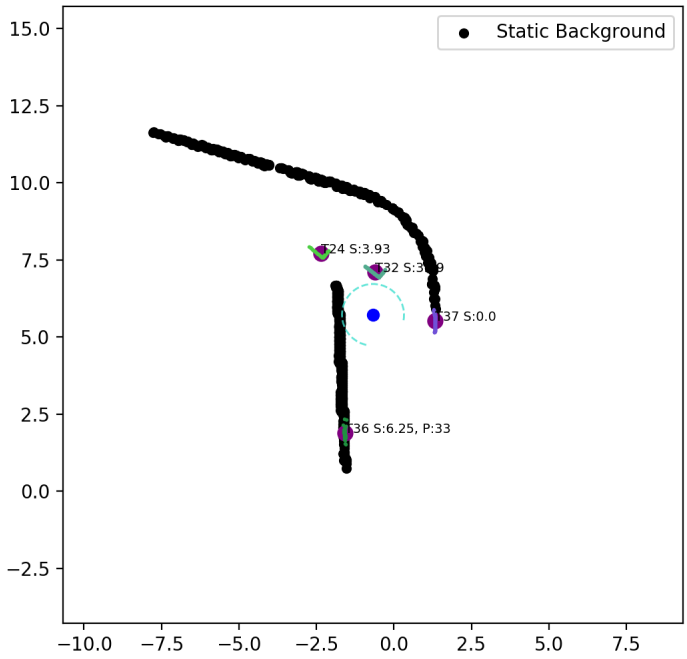


Figure 9: Sample plotted output of algorithm. In this map, the ego vehicle is denoted by the blue dot in the center; and it is tracking two cars, denoted as T24 (Track 24), and T32 (Track 32). The black dots surrounding the objects are the static walls. The estimated speed of the tracked dynamic objects is denoted in the S: parameter, in meters per second.

7.2 Runtime

The runtime of the algorithm is shown in Figure 10. The algorithm runs at an average of 5Hz on a relatively weak computer (a 1.4GHz Quad Core MacBook Pro), a relatively fast rate and an improvement on the 3Hz measured in the original proposed system [5]. The system performance is heavily dependent on the complexity of the environment: environments with complex geometries cannot be simplified well with our RDP algorithm, and thus end up sending lots of points into the JCBB, slowing down the system.

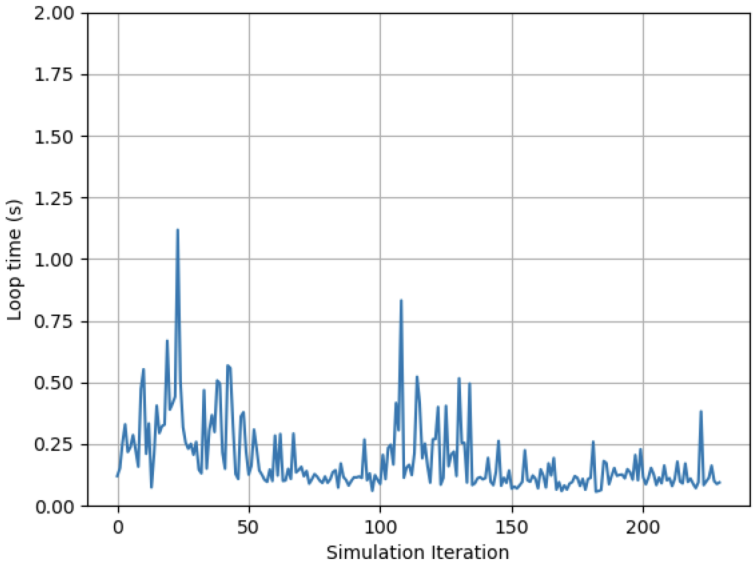


Figure 10: Total algorithm runtime (without plotting) per LiDAR update loop on a MacBook Pro with a 1.4 GHz Quad-Core processor. Certain loops where the ego vehicle encounters complex geometry(such as a long, jagged wall that cannot be much simplified with RDP) generate larger runtimes. However, the average runtime for the entire algorithm is 0.187 seconds; more than 5Hz.

7.3 Object Tracking Duration

Figure 11 shows how long are we able to maintain track of specific object in the first 500 iterations. We can see that at the beginning, the system does not track the object immediately, because the tracked object pauses for a few frames (and thus is associated with the static background), before it starts actually moving. Later on, at frame 27, for 3 consecutive frames, the system again marks the object as static. This second marking, however, is erroneous –it is due to the fact that the car is moving at a slow (but non–zero) speed.

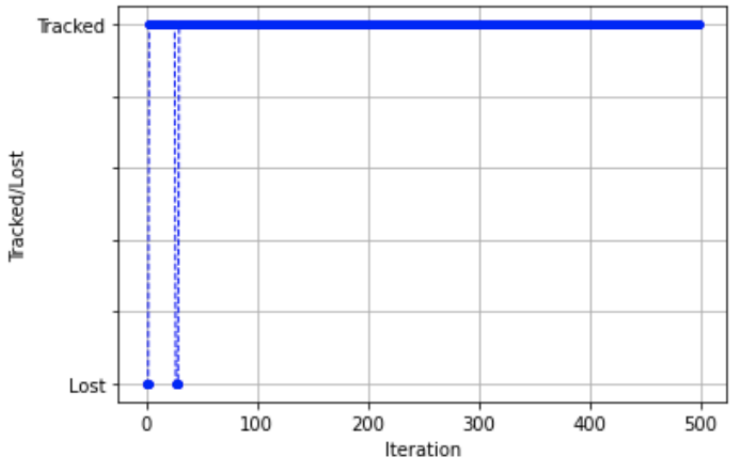


Figure 11: Duration in terms of number of frames of tracking of one specific dynamic object. After the first few frames, we are able to keep consistent track of the dynamic object apart from 3 frames at around frame 30. In these frames, the algorithm incorrectly marks the slow–moving object as part of the static background due to its low velocity.

7.4 Object Tracking Precision

Figure 12 shows the performance of our algorithm in correctly estimating the velocity of a tracked car. As can be seen, while the system velocity estimate diverges at the first few time steps, we quickly converge to within 95% of the ground truth velocity. At some points, however, we lose the target, as indicated by when the output velocity hits 0.

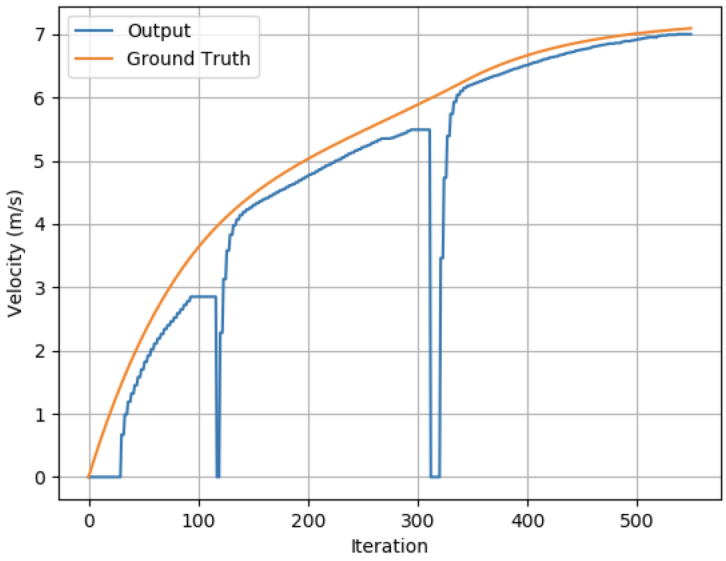


Figure 12: Velocity tracking error of the system. After the first few timesteps, we consistently stay within 95% of the true velocity, save for a few moments when we lose the tracked vehicle.

8 Conclusions and Discussion

We have demonstrated that our algorithm can run in F1Tenth simulator with one tracked car relatively smoothly. Furthermore, we have demonstrated that our modifications to the original system proposed in [5] help improve the tracking robustness and speed of the overall system. Given the restricted time frame and limitations given by pandemic and remote cooperation, we were able to achieve a working solution of dynamic object detection and tracking. However, there are still some things we want to improve. Firstly, we want to focus on improving tracking robustness, meaning that the car should be able to track targets for longer more reliably without losing them, particularly when multiple objects are next to each other. Next, we would like to even further improve the speed of the system, especially of plotting as right now the plots and the systems are using same thread which creates a bottleneck for real-time speed evaluation. This would allow us to test the algorithm on maximum velocities and make it usable for the goal of F1Tenth which is racing.

Our final goal was to implement object detection and tracking algorithm using only 2D LiDAR sensor. The most challenging part proven to be having efficiently running solution of combination of different small algorithms working reliably together to achieve the final result. Although our initial system ran quite slowly, we were able to deliver significant speed improvements after several weeks of intensive work and changes made to the system originally proposed by [5].

8.1 Immediate Next Steps

Testing on actual car: Due to the limitations of the pandemic, we have been unable to test our test our system on the actual F1Tenth racing vehicle. If possible, during the summer when things open up more, we hope to test our system ourselves, or have someone else on the F1Tenth team test it for us. **Algorithm**

speed: As mentioned we are working on improving the speed of plotting by plotting on different thread, but due some of the internal functions of the F1tenth gym simulation environment using OpenGL, this turned out to be a non-trivial task. We wish to explore two avenues to speed up plotting: moving to ROS, or using PyQTGraph. Moving to a framework like ROS, which would allow us to spin up a plotting node that runs on a separate thread, would allow us to significantly speed up our system. Our system currently is designed to be easily portable to ROS: we've intentionally limited the access points of our system to one script (tracker-gym.py), users wishing to port our system to ROS would only need to make small changes to that file. We currently have a version of the system working in PyQTGraph which enables faster plotting–however, further UI work needs to be done to make the plot readable.

Merging: Next, we want to improve the performance of our track merging algorithm. Currently, tracks are lost somewhat frequently and not merged with the tracks that replace them, due in large part to filter smugness. We want to experiment with improving the merging code, or adding more dynamics propagation noise to track velocity estimates to help solve the filter smugness issue.

Further testing: Finally, we want to test the system in different maps and adjusting parameters accordingly to achieve best results overall, rather than overfitting for one certain scenario.

Expanding classifications outside of static and dynamic binary: Figure 11 gives an example of a scenario where a tracked car that is sitting idle is associated with the static background. However, an autonomous car would presumably like to know that this object is **capable** of moving. For example, if the tracked object was a stopped car that is about to move again, the autonomous car would presumably want to track this object as a potentially dynamic object, and plan accordingly.

This could invite a fusion of camera and LiDAR data. If a camera can identify **what** an object is (whether it is a dumpster, a car, etc), then our LiDAR system can identify whether or not it is a potentially dynamic object. Future work could look into such camera+2D LiDAR combinations.

8.2 Further Work

The plan is to extend this system by collision detection and warning of approximate objects danger, which would allow us to deploy it on remotely controlled vehicles as safety feature for the operator, while not affecting performance or trying to take over the control of vehicle. This work will be done in collaboration with research group Smart Mobility Systems of Berlin Institute of Technology.

References

[1] A. Mendes, L. Conde Bento, and U. Nunes, "Multi-target detection and tracking with a laser scanner," 07 2004, pp. 796 – 801.

[2] H. Zhao, X. Shao, K. Katabira, and R. Shibasaki, "Joint tracking and classification of moving objects at intersection using a single-row laser range scanner," 10 2006, pp. 287 – 294.

[3] M. Lindstrom and J.-O. Eklundh, "Detecting and tracking moving objects from a mobile platform using a laser range scanner," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, vol. 3, 2001, pp. 1364–1369 vol.3.

[4] A. Petrovskaya, "Vehicle detection and tracking," *Stanford University Thesis*, 2011.

[5] I. P. Dominic Zeng Wang and P. Newman, "Model-free detection and tracking of dynamic objects with 2d lidar," *The International Journal of Robotics Research 2015, Vol. 34(7) 1039-1063*, 2015.

[6] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 2, pp. 239-256, doi: 10.1109/34.121791.*, Feb. 1992.

[7] W. contributors, "Ramer–douglas–peucker algorithm," https://en.wikipedia.org/wiki/Ramer-Douglas-Peucker_algorithm, online; accessed 12-May-2021.

[8] B. Farjad, A. Gupta, H. Sartipizadeh, and A. Cannon, "A novel approach for selecting extreme climate change scenarios for climate change impact studies," *Science of The Total Environment*, vol. 678, 04 2019.

[9] D. P. H. Pedro F. Felzenszwalb, "Efficient graph-based image segmentation," *International Journal of Computer Vision 2004, Vol. 59 167-181*, 2004.

[10] W. contributors, "Iterative closest point," https://en.wikipedia.org/wiki/Iterative_closest_point, online; accessed 12-May-2021.

[11] C. DâSouza, "Fundamentals of kalman filtering and estimation in aerospace engineering," https://nescacademy.nasa.gov/review/downloadfile.php?file=FundamentalsofKalmanFiltering_Presentation.pdf&id=199&distr=Public.