ECE6460 Final Project
Lidar/Radar Fusion & Classification

Alex Tyshka
Matthew Bellafaire
Pourya Shahverdi

Fall 2022

Contents

1 Introduction

The goal of this project was to create a 3D object tracking system for autonomous vehicles that could simultaneously fuse data from the LIDAR and radar sensors of the vehicle, merging measurements into single object tracks. LIDAR provides a wide short-range view of the forward environment, while radar provides longer-range data that is also capable of penetrating many objects. By combining the LIDAR and radar measurements, the vehicle can detect objects over a greater field and with greater tracking accuracy. On top of this object tracking, YOLO image classification is employed to provide annotation of objects in the environment, which could be used to inform driving decisions by an autonomous vehicle.

2 Development Process

The basis for this project started with the Homework 3 & 4 nodes, which implemented LIDAR object detection and tracking. From this point, radar boxes and YOLO tracks could be incorporated through additions to the main node and Kalman Filter. However, some issues were already appearing when running the LIDAR-only system on the sample bag. Namely, the approach worked

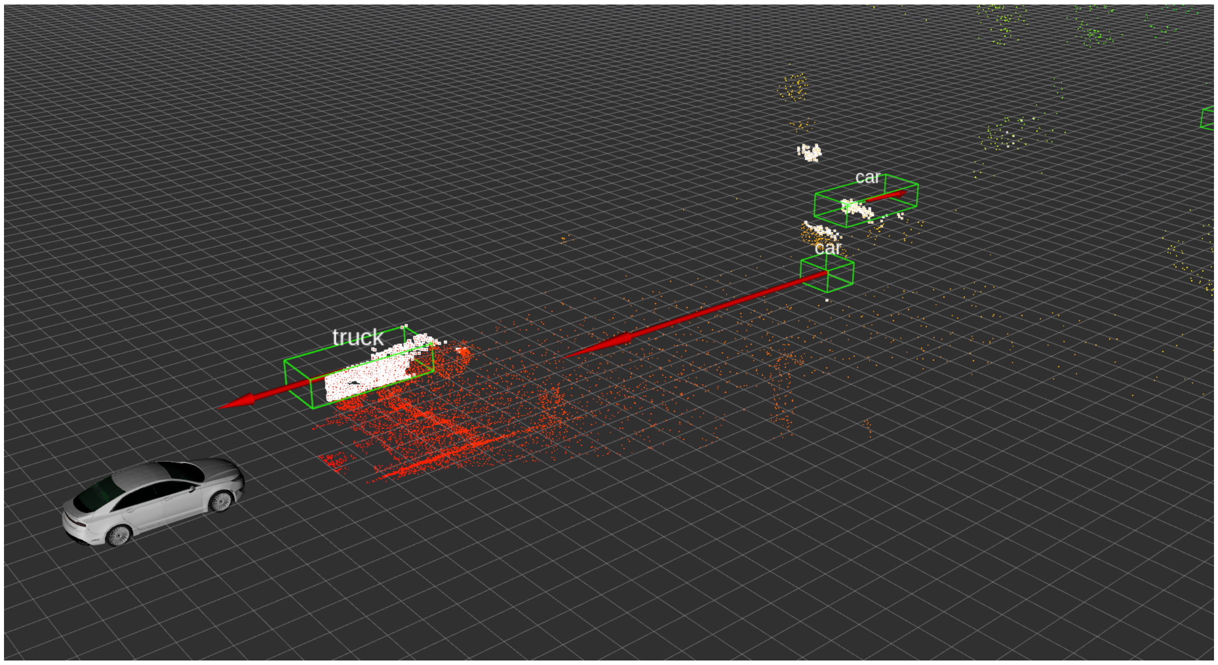

Figure 1: Output of the system described in this document in RVIZ

fairly well for objects on the road with predictable structures, like cars. However, off-road objects like trees and guardrails generated erratic box detections which proved useless for tracking. At this point, the team decided to pursue an approach to only track the dynamic objects in the scene, i.e., cars and trucks.

Because LIDAR cannot directly measure velocities, radar was used as the base sensor for tracking, with LIDAR used to refine the measurements from the radar. Relative velocities from radar were compensated with the vehicle twist to obtain absolute velocities, from which static objects could be filtered. However, this approach did not work well for all objects. LIDAR boxes for vehicles would sometimes be split or merged together by Euclidean clustering, which confused the Kalman filter greatly. To resolve this problem, the second version of LIDAR fusion was implemented, which sampled the area around existing EKF boxes to find points, which could then be used to update the EKF instance. This approach required some tuning for ground plane removal, timing uncertainty, and sensor noise estimation, but eventually, the results outperformed the initial approach.

The resulting radar-LIDAR fusion worked well for tracking vehicles in motion, but when slowing down at a stoplight, tracking would stop because radar measurements with no velocity were filtered out. To solve this challenge, YOLO object detections were used to include static radar data if it overlapped with an object detected by the camera. Additionally, all objects visible to the camera were tagged with a label that could be visualized. The resulting system is the final version of the project.

3 Implementation

An overview of the system created for this project is shown in Figure 2. All of the code written for this project is encapsulated under the multi_object_tracker, which handles the segmentation of LIDAR, processing of radar inputs, YOLO object matching, and assignment of measurements to object trackers. The multi_object_tracker node was originally derived from the code provided in

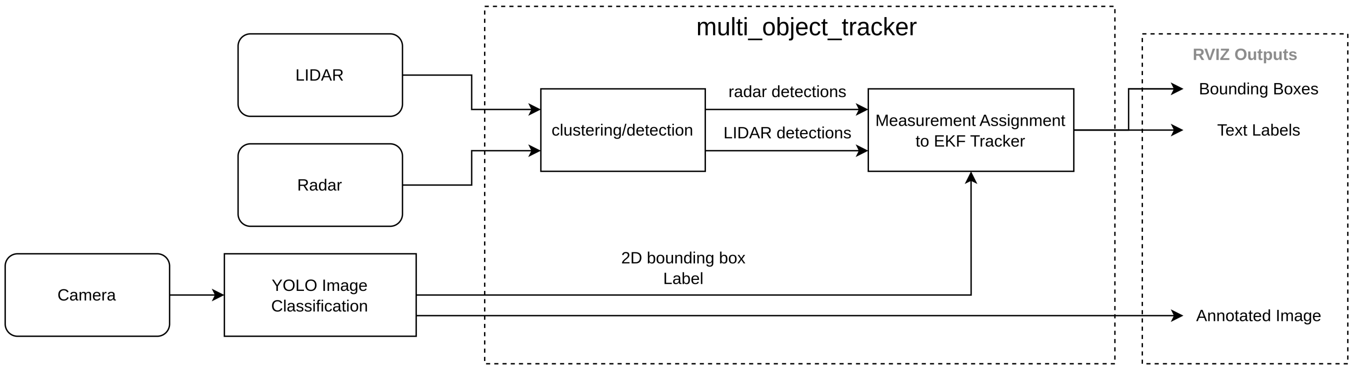

Figure 2: overview of the system implementation

Homework 4, modifications were first made to add radar measurements into the object tracker.

In the same way as Homework 4, an ObjectEKF object is used in order to track each individual object with an Extended Kalman Filter. To facilitate the fusion of radar and LIDAR information the samples from both sensors needed to be transformed into the vehicle frame through a TF transform. With both the radar and LIDAR measurements placed in the same frame, they can be passed into their respective update functions for the ObjectEKF. In the case of LIDAR the ObjectEKF will only update the xy position of the object, whereas radar provides xy position and velocity.

3.1 ObjectEKF

The ObjectEKF object was modified from the Homework 4 files. This class is responsible for tracking individual objects, each object contains its own Extended Kalman Filter (EKF) to estimate the object state. For this project, the state variables for the ObjectEKF are the same as Homework 4, tracking xy position and velocity as shown in Equation 1.

$$X_k = \begin{bmatrix} x & \dot{x} & y & \dot{y} \end{bmatrix}^T \tag{1}$$

To improve the performance of the filter, the state transition function was modified to account for the turning motion of the vehicle. When the vehicle turns, objects in the vehicle's view move with high velocity relative to the vehicle, depending on their distance from the vehicle. Without accounting for the vehicle turning, the filter would need to estimate the position of the objects based on the updated measurements, which could lead to high velocities or measurements. To remedy this, the state transition function was modified to compensate for turning, as shown in Equation 2. The additions to the state transition function were derived as the time derivative of an inverse rotation matrix where the rotation amount is the yaw rate multiplied by the time step. The state Jacobian matrix was also recomputed to align with these changes as shown in Equation 3.

$$X_{k|k-1} = \begin{bmatrix} x + \dot{x}t + t(-\dot{\psi}x\sin(\dot{\psi}t) + \dot{\psi}y\cos(\dot{\psi}t)) \\ \dot{x} \\ y + \dot{y}t + t(-\dot{\psi}x\cos(\dot{\psi}t) - \dot{\psi}y\sin(\dot{\psi}t)) \\ \dot{y} \end{bmatrix} \tag{2}$$

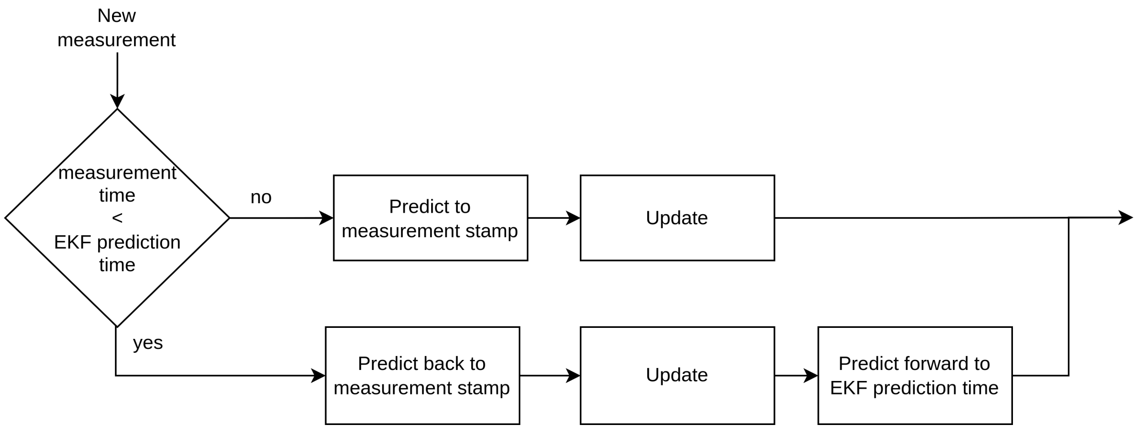
New measurement
measurement time < EKF prediction time
no
yes
Predict to measurement stamp
Update
Predict back to measurement stamp
Update
Predict forward to EKF prediction time

Figure 3: Method for updating filter with measurements stamped before the current filter time

$$A_k = \frac{\partial f}{\partial X} = \begin{bmatrix} -\dot{\psi}t\sin(-\dot{\psi}t)+1 & t & \dot{\psi}t\cos(-\dot{\psi}t) & 0 \\ 0 & 1 & 0 & 0 \\ -\dot{\psi}t\cos(-\dot{\psi}t) & 0 & -\dot{\psi}t\sin(-\dot{\psi}t)+1 & t \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

The ObjectEKF was modified to support measurements from both radar and LIDAR information. Radar information provides xy position and velocity at distances beyond what LIDAR is capable of. However, the LIDAR information is still useful in tracking nearby objects, therefore supporting both provides more accurate tracking of external objects. When building this system, we found that LIDAR objects at further distances tended to give more noisy position readings. To compensate for this behavior, we modified the measurement noise matrix for the LIDAR measurements as shown in Equation 4. In Equation 4, n represents the number of LIDAR points in a given object measurement, and r_{lidar} is the noise value configured by the dynamic reconfigure interface. With this approach, LIDAR objects that are near to the vehicle, and therefore have many points, are preferred by the filter more than those which are further away.

$$R_{lidar} = \begin{bmatrix} \frac{r_{lidar}}{\lceil n/100 \rceil} & 0 \\ 0 & \frac{r_{lidar}}{\lceil n/100 \rceil} \end{bmatrix} \tag{4}$$

The final modification that was made to the ObjectEKF was made to handle time differences in the measurements from LIDAR and radar. It was found that the radar measurements were often time-stamped behind the current time of the filter, in order to create stable estimates of the objects incorporating both LIDAR and radar, this had to be compensated for. In cases where a measurement timestamp is after the current time of the filter, the update function is run normally. However, in cases where the filter is ahead of a measurement, the filter must first run the prediction step backwards to the time of the measurement, then run the update function, and then run the prediction step back forward. Figure 3 shows the flow of this process.

3.2 LIDAR Sampling

To update the EKF with LIDAR data, the input pointcloud is first transformed into the base frame. Next, the EKF objects are projected to the current timestamp of the pointcloud for optimal correspondence. However, these EKFs will not align perfectly, mainly because the radar data is

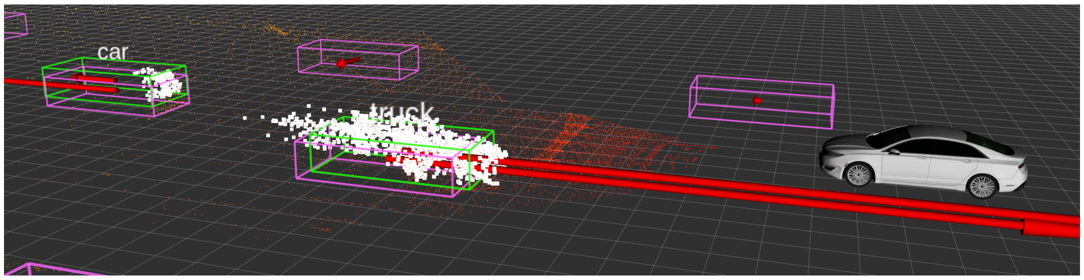

Figure 4: LIDAR points shown in white are associated with objects

not accurately timestamped; this discrepancy increases with the velocity of the object. To sample LIDAR points, a CropBox filter is used to sample the region around an oriented box. To account for the mismatch with radar, the sampling box is scaled along the x-axis by the object velocity so that if the LIDAR object is slightly leading or trailing the radar object it will still be detected.

Next, the ground plane must be removed from the sampled pointcloud. For small pointclouds, (n <100 points), the ground plane is minimal, so this step is skipped. Otherwise, a priority queue is utilized to find the lowest 25% of points. From this, a z threshold is set at 25cm above the 25% mark, which is used to filter the object points. This approach yielded better results and runs faster than a normals-based filter. The remaining points are used to compute a centroid, which is input to the ObjectEKF. Ideally the LIDAR data would also update the box dimensions, however, there was not sufficient time to implement a working version of this goal.

Listing 1: Ground Plane Removal

```
std::priority_queue<float> min_points;
int queue_size = tmp_indices.size() / 4;
for (auto index : tmp_indices)
{
  float z = cloud_in->points[index].z;
  if (min_points.size() < queue_size)
  {
    min_points.push(z);
  } else if (cloud_in->points[index].z < min_points.top())
  {
    min_points.push(z);
    min_points.pop();
  }
}
float min_z = min_points.top();
tmp_indices.erase(std::remove_if(tmp_indices.begin(), tmp_indices.end(),
                              [&cloud_in, min_z](int i){
  return cloud_in->points[i].z < min_z + 0.25;
}), tmp_indices.end());
```

3.3 Camera Detection

The camera detection system subscribes to the output of the YOLO neural network and associates its object detections with the radar data and EKF instances. To accomplish this, the 3D EKF boxes are projected to 2D boxes using the method from the sample sensor fusion code. These boxes are compared to the YOLO boxes using the Intersection over Union (IoU) metric, a common method for object detection. If the best match has at least 0.3 IoU, it is considered a match, and the label

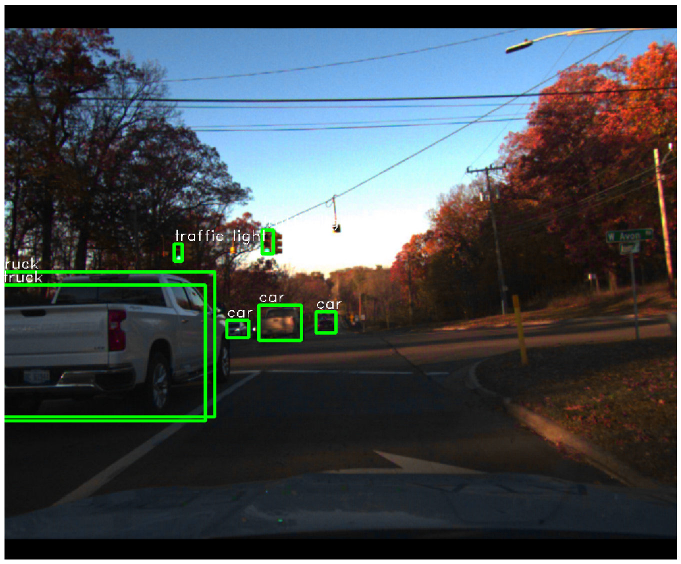

Figure 5: Object classification in camera view

is assigned to the EKF instance. This label persists even if the object moves out of camera view but remains tracked by the radar.

3.4 Multi Object Tracker

The multi-object tracker subscribes to the radar, LIDAR, and camera data and determines how to update the EKF instances accordingly. When a radar message is received, the node first transforms the boxes and removes those outside the configured Y window. Next, the relative velocity of the object is converted to an absolute velocity using the following equation:

$$V_{obj,absolute} = V_{obj,relative} + V_{vehicle,linear} + P_{obj} \times V_{vehicle,angular} \tag{5}$$

Any radar object with low compensated velocity is marked as static and will not be used for tracking unless it matches a YOLO detection with sufficient IoU, in which case it will not be ignored. From there, the tracker follows the same Euclidean-distance matching algorithm as Homework 4 for updating the EKFs.

4 Conclusion

Overall, this project successfully implemented an approach for combining radar, LIDAR, and camera data on real-world sensor data. There were significant challenges, namely the noisy bounding boxes and lack of precision radar time stamping, but the EKF managed to successfully account for the noise and uncertainties of the data and produce smooth tracking. Some false positive tracks still occur, but these are significantly reduced compared to the raw data.

In future work, the accuracy of the system could be improved in several ways. Neural networks are state-of-the-art for object detection and perform much better in noisy environments than hand-crafted detection approaches; employing a multi-modal neural network for camera and LIDAR data could significantly improve the quality of LIDAR detections. Additionally, receiving accurately timestamped radar data would significantly improve detection quality, and receiving raw radar pointclouds would allow for even deeper sensor fusion.