Obstacle Detection and Tracking in an Urban Environment Using 3D LiDAR and a Mobileye 560

by Veronica M. Lane

S.B., E.E.C.S. M.I.T., 2015

Submitted to the

Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June 2017

Author: ____

Department of Electrical Engineering and Computer Science

July 26, 2017

Certified by: ____

Sertac Karaman, Associate Professor of Aeronautics and Astronautics, Thesis Supervisor

July 26, 2017

Accepted by: ____

Christopher Terman, Chairman, Masters of Engineering Thesis Committee

Obstacle Detection and Tracking in an Urban Environment Using 3D LiDAR and a Mobileye 560

by Veronica M. Lane

Submitted to the Department of Electrical Engineering and Computer Science

July 26, 2017

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

In order to navigate in an urban environment, a vehicle must be able to reliably detect and track dynamic obstacles such as vehicles, pedestrians, bicycles, and motorcycles. This paper presents a sensor fusion algorithm which combines tracking information from a Mobileye 560 and a Velodyne HDL-64E. The Velodyne tracking module first extracts obstacles by removing the ground plane points and then segmenting the remaining points using Euclidean Cluster Extraction. The Velodyne tracking module then uses the Kuhn-Munkres algorithm to associate Velodyne obstacles of the same type between time steps. The sensor fusion module associates and tracks obstacles from both the Velodyne and Mobileye tracking modules. It is able to reliably associate the same Velodyne and Mobileye obstacle between frames, although the Velodyne tracking module only provides robust tracking in simple scenes such as bridges.

Chapter 1

Introduction

Autonomous vehicle technology will enable people and goods to move more safely and efficiently without the need for human operation. New driver assistance systems and fully autonomous systems are currently being developed by commercial companies and academic institutions to navigate in urban environments, which are uniquely challenging because they are more unstructured than highways. For example, urban traffic is more heterogeneous and an autonomous vehicle must be able to detect and track bicycles and pedestrians in addition to vehicles, trucks, and motorcycles. To safely navigate through intersections, a vehicle must be able to track oncoming and crossing traffic and predict their movements. For instance, the vehicle must be able to predict if an oncoming vehicle plans to make an unprotected left turn. The vehicle must be able to detect pedestrians and predict whether or not they will cross the intersection. Furthermore, different types of traffic obstacles behave very differently, such as city buses, which stop much more often than other vehicles.

A multi-sensor approach for tracking dynamic obstacles is more reliable than a single-sensor approach, but also adds complexity to the system. Different types of sensors can provide complementary information about the scene. For example, the Velodyne HDL-64E sensor provides a 3D map of the environment with a 360 field of view. It can detect passing, oncoming, and crossing traffic. However, while the Velodyne provides more detailed range information and a wider field of view than a camera, it costs about $\$75,000$, its range is more limited than a camera, and the refresh rate (about 10 Hz) is lower. Furthermore, it is unreliable in adverse weather conditions such as snow, fog, and rain. On the other hand, since a camera provides RGB information it can better classify objects in the scene such as vehicles and pedestrians.

Recently, a few companies, including Mobileye, have marketed vision-based advanced driver-assistance systems which warn drivers of potential collisions and lane departures. The Mobileye vision systems do not provide the raw camera image, but instead provide information about the visible obstacles and lanes. The Mobileye can only be safely used on highways since it cannot detect passing, oncoming, or crossing traffic. However, in urban environments, the Mobileye is still able to detect some obstacles, particularly the closest in path vehicle. Tesla used the Mobileye technology in their Model S car to allow the hands-off driving on freeways. However, the driver was expected monitor the vehicle and be ready to intervene at all times.

The Mobileye enables a vehicle to drive semi-autonomously. Fusing Mobileye data with information from a Velodyne HDL-64E can enable a vehicle to better track the obstacles in its environment, since the Velodyne can detect passing, oncoming and crossing traffic. This paper describes an architecture for sensor data fusion of an off-the-shelf vision system, the Mobileye 560, and a Velodyne HDL-64E to enable a vehicle to track dynamic obstacles in an urban environment, as well as the systems integration of the Mobileye 560. Both sensors have a specific module which track detected objects. The tracking data is then fused together to provide a list of obstacles in the scene. The sensor fusion module reliably fuses tracked Velodyne and Mobileye obstacles, but the Velodyne tracker is only robust in simple scenes like bridges. However, the sensor fusion module is able to correctly match obstacles even when the Velodyne label for a given obstacle changes.

Chapter 2

Related Works

Darms et al. [1] presented a multi-obstacle tracking algorithm used in the DARPA Urban Grand Challenge. The algorithm combines information from 13 different sensors including planar LIDAR sensors, a Velodyne HDL-64E, radar sensors, and a fixed beam laser sensor. The tracking algorithm is switched based on available sensor information, and all obstacles are described using a box or point model. The sensor fusion architecture is separated into individual sensor layers and a fusion layer. The sensor layers extract the features, associate the features with a current obstacle, and determine the obstacle state of movement. The fusion layer selects the best model for the obstacle and determines the global movement of the obstacle. The tracking algorithm was able to reliably detect and track obstacles in the scene, but it required information from 13 different sensors. The algorithm was very computationally expensive and each additional sensor increased the cost of the vehicle.

Himmelsbach et al. [2] presented a system which classified and tracked vehicles in a 3D point cloud. The cloud is first segmented using a 2D occupancy grid. Then segmented objects are classified by passing the histograms of the point features to an SVM classifier. Finally, only obstacles classified as vehicles are tracked, and the tracked vehicle states are estimated using a multiple model Kalman Filter. Although the system was able to robustly track vehicles, it could not detect other dynamic obstacles in the scene.

Morton et al. [3] evaluated various object tracking methods with 3D LIDAR. Early methods projected the 3D point cloud onto a 2D representation and used model based tracking. A later method uses a learning classifier and multi-hypothesis tracking to detect and track pedestrians in a scene. Douillard [4] and Moosman [5] presented algorithms which are able to fully segment the point cloud. Morton proposed the Mesh-Voxel Segmentation algorithm creates a terrain mesh from the LiDAR data, extracts the ground from the computed gradient field in the terrain mesh, and clusters the ground points based on voxel grid connectivity. The method iteratively grows the ground partition proceeding from the center onwards. Once the points were segmented, Morton divided the tracking problem into four separate tasks: motion estimation, registration, appearance modeling and data association. Each obstacle is represented by a vector including the time, global position, local position in the LIDAR frame, surface normal, and ID. Each tracking method uses a motion model to predict the likely future location and matches the new observation with the current tracked objects. It also adds objects which are not currently being tracked. The simplest tracking model tested used the newly observed centroid to update the tracked position. Morton tested this tracking method both with and without a Kalman filter, using a constant velocity motion model. Another tracking method aligned each tracked object individually using ICP. This method used appearance models to track objects and combined the appearance model with new observations and filtered based on uncertainty. However, this caused blurring over time, and methods which gradually replaced points performed better. Morton's analysis showed that the ICP based tracking methods did not have significant advantages over the simple centroid based tracking method, which is used in this paper. The sensor fusion algorithm in this paper uses simple tracking model presented by Morton, which tracks obstacle centroids and associates obstacles with the Kuhn-Munkres Algorithm [6].

Chavez-Garcia et al. [7] proposed a sensor fusion architecture for tracking obstacles which fuses data from radar, 3D LiDAR, and a camera. They use a 2D occupancy grid map to represent the LIDAR data. If a previously unoccupied cell is marked as occupied, the cell belongs to a moving object. Before fusing the data, the camera module first detects and classifies the objects as either a pedestrian, bicycle, car, or truck. The sensor data is fused using an evidential framework.

Zhang et al. [8] propose a vehicle tracking method which uses a Velodyne HDL-64E and combines a Multiple Hypothesis (MHT) with Dynamic Point Cloud Registration (DPCR). The method does not need the vehicle's GPS or IMU to track the targets, instead the DPCR is used to estimate the vehicle pose. The tracking method is able to track obstacles which become occluded between frames. Although this method provides robust tracking of vehicles, it cannot track other dynamic obstacles.

Mirzaei et al. [9] describe a 3D LiDAR and camera intrinsic and extrinsic calibration method. The algorithm estimates unknown parameters by dividing the estimation problem into two least-squares sub-problems. The estimates are improved through batch nonlinear least squares minimization. However, this calibration method cannot be used for the Mobileye and the Velodyne since the camera intrinsics are unknown and the Mobileye camera image cannot be accessed.

Chapter 3

System Description

3.1 The Vehicle

The vehicle used for testing is a 2015 Toyota Prius V, shown in Figure 3.1.1. The vehicle's brake pedal, gear selector, and steering column are controlled by electric motors which can be controlled by the vehicles Controller Area Network (CAN). The CAN bus allows vehicle devices to communicate with each other without a host computer.

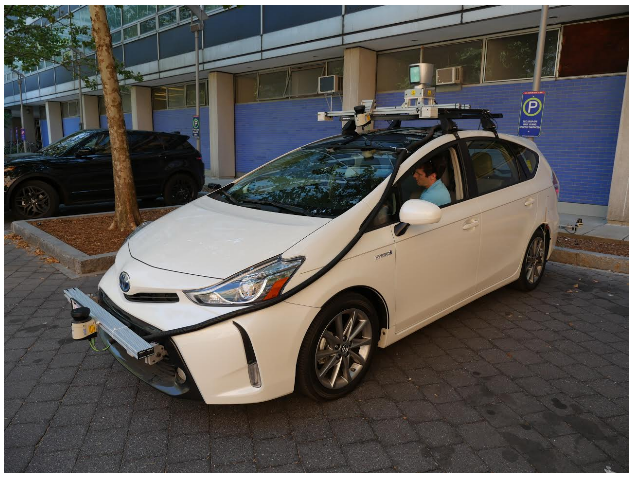
A combination of 10 sensors are integrated into the vehicle. A Velodyne HDL-64E S3 is mounted on top of the vehicle. Additionally, a Mobileye 560 and Point Grey Ladybug 3 camera are mounted on the front windshield. Two SICK TIM551 sensors are mounted on the left and right of the back bumper. Furthermore, SICK LMS151 sensors are mounted on the front bumper and on the top of the vehicle in the front. Finally, the vehicle is also equipped with a LORD Microstrain 3DM-GX4-25 AHRS, and encoders mounted on the back wheels.

3.2 Sensor Descriptions

3.2.1 Velodyne-64E

The Velodyne HDL-64E S3, shown in Figure 3.2.1, is a high definition LIDAR sensor which provides a real-time 3D point cloud of the surrounding environment. The sensor uses a rotating head and 64 LiDAR channels aligned between $2.0°$ and $-24.9°$, for a total vertical field of view of $26.9°$. It has a $360°$ horizontal field of view and a range of up to 120 m. The Velodyne can spin at a rate between 5 Hz and 20 Hz. As the spin rate increases, the resolution decreases. We set the spin rate to 10 Hz. The Velodyne detects

Figure 3.1.1: The Toyota Prius V



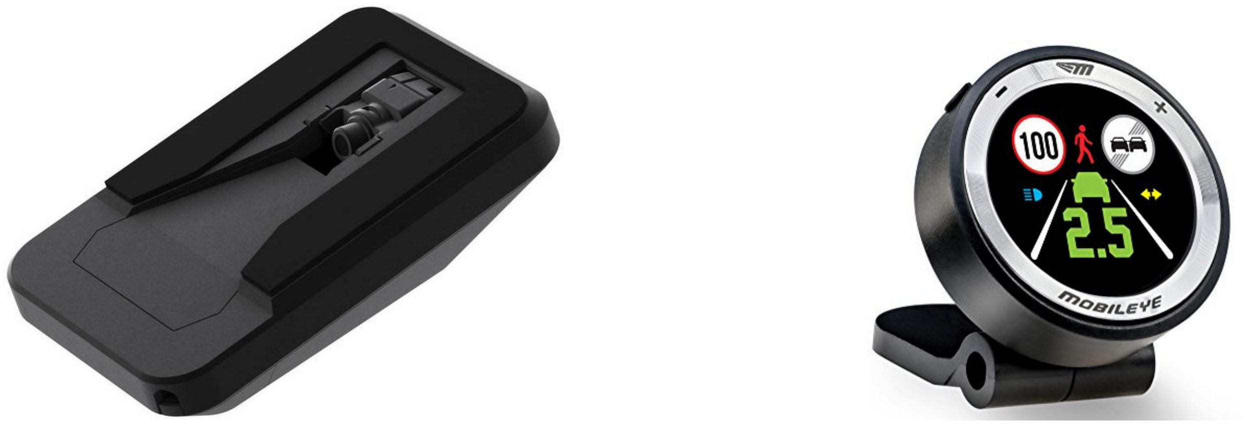
approximately 1.3 million points per second. We used the ROS Velodyne packages to collect data from the sensor [10].

Figure 3.2.1: The Velodyne HDL-64E [11]

3.2.2 Mobileye 560

The Mobileye 560, shown in Figure 3.2.2a, is an Advanced Driver Assistance System (ADAS), which includes a High Dynamic Range CMOS (HDRC) camera, an image processing board, and a display for visual warnings to the driver as shown in Figure 3.2.2b. The camera is mounted on the front windshield and the system communicates via the a vehicle's CAN. The Mobileye transmits a series of messages every 66-100 ms on a 500 kbps channel.

The Mobileye does not transmit the raw camera image, but instead transmits information about the dynamic obstacles, lanes, and traffic signs around the vehicle, as well as collision warnings, lane departure warnings, headway monitoring, and intelligent high beam control decisions. The Mobileye 5 series is developed for paved roads with clear lane markings. It can operate at night and during adverse weather conditions, but the recognition is not guaranteed to be as robust. It can only reliably detect the fully visible rear ends of vehicles and bicycles as well as fully visible pedestrians. It does not provide robust detection of crossing, oncoming, and passing traffic.



(a) Mobileye Sensor [12] (b) Mobileye Display [13]

Figure 3.2.2: The Mobileye System

Kvaser Leaf Light v2

To communicate with the Mobileye we used a Kvaser Leaf Light v2 which connects a computer to the CAN bus via a USB port. Kvaser provides a Linux library with device drivers as well SDKs written in C and Python, which provide methods for CAN channel configuration and communication.

When starting up the vehicle, we consistently had issues connecting to the Leaf Light. Each user had a laptop with with Ubuntu 16.04 and the Kvaser Linux library installed. During installation the Kvaser library did not sign the kernel modules for the device drivers. When UEFI secure boot is enabled, Linux kernel versions 4.4.0-20 onwards will only load kernel modules which have been digitally signed with the proper key. Since Kvaser did not sign the module, users either had to manually sign the module or disable UEFI secure boot to load the Leaf Light kernel module.

Even with signed kernel modules or secure boot disabled, we found that each of our systems would often fail to recognize the Kvaser device. Common troubleshooting techniques, such as unplugging and replugging the device and restarting the computer failed to solve the problem. However, removing and reinstalling the Kvaser library (then signing the kernel if necessary) and restarting the computer solved the problem. This troubleshooting process was often needed when first connecting to the device. However, once the Leaf Light connected to a laptop, it worked consistently without issues.

Chapter 4

Approach

The sensor fusion system fuses data from the Velodyne and the Mobileye to track obstacles in the environment. An overview of the architecture is shown in Figure 4.0.1. Each sensor has its own module which tracks the detected obstacles. The Mobileye Node processes the CAN data and keeps track of the obstacles reported by the Mobileye. The Velodyne Tracking Node processes the point cloud, extracts the features, and tracks the obstacles detected by the Velodyne. The Fusion Node then associates Velodyne and Mobileye obstacles.

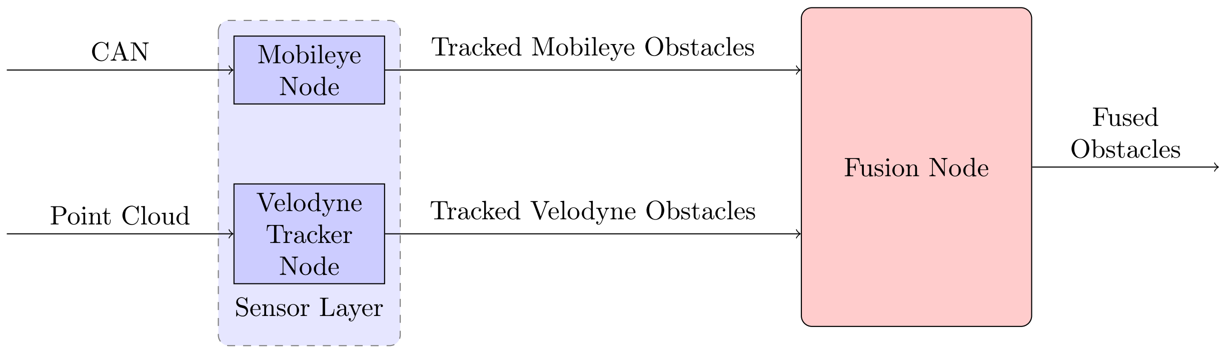


Figure 4.0.1: An overview of the sensor fusion system architecture

4.1 Sensor Calibration and Registration

4.1.1 Mobileye Calibration

Upon installation, the Mobileye must be calibrated. The user can adjust the camera angle to point directly forward. Then, the user provides the Mobileye with the camera height from the ground and must mark

the camera height on a TAC target with a line. Next, the user places the TAC target as close to the front bumper as possible and adjusts camera angle to align the marked line on the target to be between two red lines displayed on the camera image. Additionally, the user provides the Mobileye with the width of the vehicle's front wheel base, and the driving side of the road (e.g. "left" for the United Kingdom). If the car hood is visible in the image, the user also indicates the location of the highest point of the hood in the image by adjusting the position of a red line in the calibration application. Furthermore, the user provides the distance to the left and right windshield edges and distance to the front bumper. The final calibration step can be completed using a TAC target or automatic calibration during the first drive.
In addition to the guided Mobileye calibration, we measured the position of the Mobileye relative to the SICK LiDAR on the front bumper. The ROS transform tree transforms between coordinate frames which are linked together [14]. In this case, the positions of the SICK mounted on the front bumper and the Velodyne were both measured relative to the position of the vehicle's IMU. Therefore, the transform tree can be used to approximate the transform from the Mobileye to the Velodyne frame.

4.2 Mobileye Interface

Using the Kvaser library, the ROS Mobileye node sets up a CAN channel and processes incoming Mobileye messages. Each CAN message is composed of a 64-bit identification header followed by 8 data bytes. For the purposes of data collection, there are two Mobileye ROS nodes, the Raw Mobileye CAN Node which processes the CAN messages and publishes the CAN message ID and data bytes in MobileyeRaw ROS messages and the Mobileye CAN Node which processes the CAN messages and publishes the corresponding ROS message.

4.2.1 Mobileye CAN ROS Node

Upon receiving the a message, the Mobileye CAN ROS node processes the data bytes and sends the corresponding ROS message. Table 4.2.1 below shows an overview of the CAN message IDs from the Mobileye and the corresponding ROS message. Since the CAN message packets are exactly 8 data bytes, obstacle and lane marking descriptions are split into multiple messages with sequential headers. The ROS node combines the information from the lane A & B CAN messages and obstacle A, B, & C CAN messages into one ROS message, respectively. To prevent errors, upon receiving an Obstacle A (or B) or Lane Marking A CAN message the node checks the ID of the next incoming CAN message. If the following message is not the corresponding B (or C) message, the node displays an error message which includes the expected ID number and received ID number and then processes the received message.

CAN Header ID(s)	**Mobileye Message Name**	**ROS Message**
0x650	Fixed Focus of Energy (FOE)	MobileyeFixedFoe
0x669	Lane	MobileyeEgoLane
0x700	Automatic Warning System (AWS) Display	MobileyeAws
0X720 - 0x726	Traffic Sign	MobileyeTrafficSign
0x727	Vision Only Traffic Sign	MobileyeVisionOnlyTsr
0x728	Active High Beam Control (AHBC)	MobileyeAhbc
0x729	AHBC Gradual	MobileyeLights
0x737	Lane	MobileyeLane
0x738	Obstacle Status	MobileyeObstacleStatus
0x739 - 0x75F	Obstacle Data A/B/C	MobileyeObstacle
0x760	Car Info	MobileyeVehicleState
0x766 - 0x767	Lane Keeping Assistant (LKA) Left Lane A & B	MobileyeLaneMarking
0X768 - 0x769	LKA Right Lane A & B	
0x76C - 0x77A	Details Next Lane A & B	
0x76A	Reference Points	MobileyeRefPoint
0x76B	Number of Next Lanes	MobileyeNextLanes

Table 4.2.1: Mobileye CAN message protocol to ROS message protocol.

The Mobileye Obstacle and Lane ROS messages are described further in the following sections, and the other messages are described in the Appendix A.

Obstacle Messages

As stated previously, the Mobileye does not reliably detect oncoming, passing, or crossing traffic. It only reliably detects fully visible pedestrians and bicycles, trucks, motorcycles, vehicles with fully visible rear ends.

The MobileyeObstacle message describes a detected dynamic obstacle in the scene. The types of dynamic obstacles detected include vehicles, trucks, motorcycles, bicycles, and pedestrians. The message includes the obstacle's ID number, type, position relative to the reference point, relative x (forward) velocity, acceleration, width, length (if the front of the vehicle is viewable), and lane position. If the obstacle is in the ego lane, which is the vehicle's current lane, or the next lane, the message indicates whether or not the vehicle is cutting in or out of the ego lane. Furthermore, it includes the angle to the center of the obstacle and the scale change from the previous frame. If the obstacle is a vehicle it also includes information about the obstacle's blinker status, brake light status, and whether it is the closest-in-path vehicle (CIPV). Finally, it includes whether or not it was a newly detected obstacle and the age of the obstacle in number of frames.

The documentation of the Mobileye states that the Obstacle positions are relative to a reference point which is one second ahead of the vehicle, likely based on the Mobileye's estimation of the vehicle velocity based on visual odometry [15]. However, the reference point was always set to 0.0, 0.0. It is unclear whether the reference point is used, or the Mobileye failed to transmit the reference point location. We contacted Mobileye to resolve this question several times but they did not respond.

Table 4.2.2 describes each of the parameters of the MobileyeObstacle message. Some of the parameter names are different than the names used in the AutonomousStuff documentation for clarity [15].

Parameter	**Description**
uint8 id	New obstacles are given the last free used ID
uint8 type	The obstacle's classification
uint8 status	The obstacle's moving status (e.g. moving, parked)
bool isCipv	Whether or not this obstacle is the Closest in Path Vehicle (CIPV)
float32 position.x	The X (longitudinal) position of the obstacle in meters relative to the reference point. It is computed from the pixel position and detected width of the obstacle and is filtered. The error is generally below 10% or two meters (the larger of the two) in 85% of cases.
float32 position.y	The Y (lateral) position of the obstacle. The y position is computed from the pixel position and the position.x value and is filtered. The error is correlated with the x error.
float32 relVelX	The relative longitudinal velocity of the obstacle computed from the obstacle scale change in the image
float32 accelX	The longitudinal acceleration of the obstacle
float32 length	The length of the obstacle [m] (longitude axis). This is only updated for next lane vehicles which are fully visible and if the front edge of the vehicle has been identified.
float32 width	The width of the obstacle [m] (lateral axis) calculated from the pixel width and the obstacle distance. The value is filtered to avoid outliers. The expected performance within an error margin of 10% in 90% of the cases. At night, the width measured is between the vehicle's taillights.
uint8 hostLaneStatus	The obstacle state with respect to host lane (e.g. in host lane, entering host lane) based on an estimation of where the target is now relative to the lanes, its rate of change, and estimation of where it is going to be within one second. It does not distinguish between left and right lanes.
uint8 lane	The lane of the obstacle (e.g. ego lane, next lane). This is calculated from the obstacle position and the lane detection or the headway model of the vehicle. The lane assignment decision takes up to 5 frames from the first detection of the obstacle.
uint8 blinker	The blinker status (e.g. left blinker on)
bool brakeLights	Whether or not the obstacle brake lights are on
float32 angleToCenter	The angle to the center of the obstacle [deg] from the reference point. The values is negative if obstacle has moved to left.
float32 angRate	Angle rate of the center of the obstacle [deg/s]
float32 scaleChange	Scale change [pixels/s]
bool isNew	Whether or not the obstacle was first detected this frame
bool replaced	Whether or not the obstacle was replaced in this frame
uint8 age	The age of the obstacle in number of frames, remains 254 if larger

Table 4.2.2: MobileyeObstacle message parameter descriptions.

Lane Messages

The Mobileye sends four different lane messages, some of which contain redundant information. The AWS Message also warns of lane departures. The Lane Departure Warnings (LDWs) are only available when the vehicle's speed is above 55 km/h (34 mph). The MobileyeLane ROS message contains lane information and measurements for the current lane including the lane curvature and heading as well as the yaw and pitch angle of the vehicle derived from the lane measurements. It also indicates if the vehicle is in a construction area and if the lane departure warning is enabled for the left and right lane marks. The MobileyeEgoLane message describes the left and right lane markings. For both markings it includes the distance to the lane marking, the marking type, whether or not lane departure warnings are available, and a confidence value on a 0-3 scale.

The MobileyeLaneMarking message describes an individual lane marking, both for the ego lane markings and next lane markings. The message id parameter indicates whether the marking is for the current lane's markings (left or right) or the next lane's marking. The message describes the distance between the camera and the lane mark, the parameters of a cubic equation describing the shape of the lane, the width of the marking, the physical view range of the marking, and the validity of the view range. Finally, the MobileyeNextLanes message indicates the number of next lane markers which were reported.

4.3 Velodyne Feature Extraction

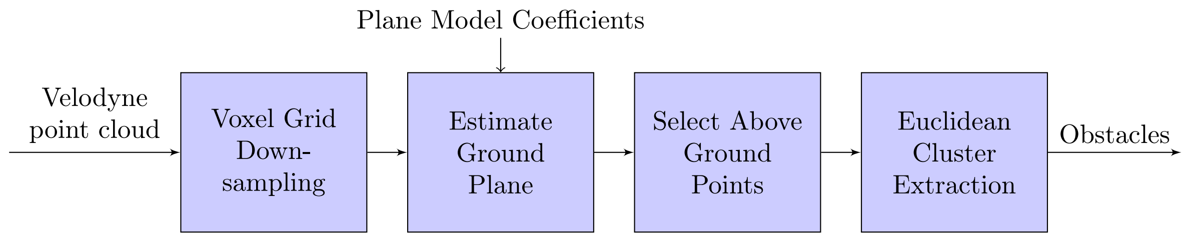


Figure 4.3.1: Velodyne Feature Extraction. First the point cloud is downsampled. Then the ground plane is estimated and the points above the ground are selected. Finally, the remaining points are clustered.

The Velodyne feature extraction module uses the Point Cloud Library (PCL) [16]. As shown in 4.3.1, first the point cloud is downsampled using a voxel grid filter then the ground plane is removed and the remaining points are clustered.

In the ground plane estimation step, first a pass through filter extracts points in the region around the ground plane, based on the z value of the points in the Velodyne frame. Then an approximate model for the ground plane is calculated. The Random Sample Consensus Algorithm (RANSAC) and the Progressive

Sample Consensus (PROSAC) algorithm were compared for extracting the ground plane. RANSAC is an iterative algorithm which uses random sampling and estimates the parameters of a mathematical model, in this case a plane, from a dataset. It is robust to outliers. However, the RANSAC method was much slower than the PROSAC method, and did not work in real time. PROSAC uses guided sampling and converges to the solution of RANSAC. It is more suited for real time applications. Furthermore, constraining the plane to be approximately parallel to the previous ground plane resulted in a better model with more inliers than a generic unconstrained plane model. A comparison of the unconstrained and constrained plane models is shown in the results section.

After the ground plane model is estimated, the points above the ground are extracted. All points which are between 50 cm and 3 meters above the ground are selected. The above ground points are passed to the cluster extraction module, which uses euclidean cluster extraction. This method uses a KD-tree to represent the point cloud. It processes each point, adding all unprocessed points within a threshold radius to the queue. Once all of the points in the queue have been processed, the points are added to the list of clusters. This process repeats until all of the points have been processed and are part of the list of clusters. Clusters with too few or too many points, based on user defined parameters, are not included in the final list of clusters.

Obstacle representations of the extracted clusters are then passed to the tracking algorithms. The obstacles passed to the tracking algorithm describe the time of detection, the x, y, z position of the cluster centroid, the standard deviations of the x, y, and z coordinates of the points in the cluster, and a bounding box representing the height, width, and length of the cluster calculated from the minimum and maximum x, y, and z coordinates of the cluster.

4.4 Velodyne Tracking

The segmented clusters are passed to the tracking algorithm. The tracking problem is split into three subproblems, data association, registration, and motion estimation.

4.4.1 Obstacle Representation

The obstacles position is estimated to be the cluster centroid and the size is estimated by a bounding box. Each new obstacle is assigned a unique ID number between 0 and 200. New obstacles are assigned the last free used ID. Since ID numbers are reused, a boolean value indicates whether or not the obstacle was replaced in a given frame. The obstacle object also indicates the time of the first and last detection. The initial estimate of the relative velocity is set to 0.

4.4.2 Data Association

In the data association step, obstacles from the current time step are matched to obstacles from the previous time step. If a tracked obstacle from the previous time step is not matched in the current frame, it is removed from the list of tracked obstacles. The data association step is treated as a linear assignment problem, a the Kuhn-Munkres algorithm is used to match obstacles. The cost of an assignment is the distance between a current obstacle's position and the interpolated position of an object from the previous time step. If the distance between two matched is above a set threshold value, the obstacles are not considered a match, unless the previous obstacle is from the first detection, in which case its position estimate is unreliable.

Velodyne obstacles are represented by three models, an box model, a point model, and line model. Obstacles are modeled as points if the x and y dimensions are below a threshold value and the standard deviation of the x and y coordinates of the cluster points are below a threshold value. Very large obstacles and long line segments are ignored, but short line segments are occasionally vehicles or bicycles, so they are tracked as well. Boxes, points, and lines are tracked and matched separately.

If the estimated position an obstacle from the previous time step is outside of the boundary box of the current clusters, it is not matched with the new list of obstacles. This prevents obstacles which move beyond the Velodyne's field of view from matching with obstacles just entering the Velodyne's field of view.

4.4.3 Registration

Currently, no filtering is applied after data association. The registration step simply updates the vehicle's current position and size with the current position and size of the latest update.

4.4.4 Motion Estimation

The motion model assumes that the velocity of an obstacle is constant between frames. The relative velocity of a tracked obstacle is estimated from the change in position of the obstacle between two time steps. The position, r of an obstacle at a given time, t, is estimated from the relative velocity through linear interpolation, $r(t) = r(t_0) + v_{\mathrm{rel}} * (t - t_0)$.

4.4.5 ROS Publishing

The tracked obstacles in a given time step are published in a single ROS message, the VelodyneObstacleStatus message. This message includes the number of tracked obstacles and a vector of VelodyneObstacles describing the unique ID, model type, position, size, relative velocity, time of first detection, and whether or not the obstacle is new.

4.5 Sensor Fusion

The Sensor Fusion node matches detected Mobileye Obstacles with detected Velodyne Obstacles. Both sensor modules track detected obstacles across frames independently. The Sensor Fusion node receives Obstacle messages and Obstacle Status messages from the Mobileye node and Velodyne Obstacle Status Messages from the Velodyne node, which describe all of the detected obstacles from a Velodyne point cloud.

The fusion node associates and registers Velodyne and Mobileye obstacles each Velodyne frame, but does not do independent tracking across frames. The Mobileye tracking is fairly unreliable in urban environments, and tracking labels frequently switch between different obstacles of the same type. Between frames, Mobileye obstacles frequently disappear and reappear, sometimes tracking the same obstacle, sometimes tracking a different one, while the old obstacle is no longer tracked. Furthermore, the Velodyne tracking is also unreliable in busy scenes, and obstacle labels are frequently changed between frames. Therefore, it cannot be assumed that a given Mobileye obstacle matched to a Velodyne obstacle in on frame will match to the same obstacle in the next frame. Furthermore, the Velodyne's field of view is 360 degrees, and it can detect obstacles which are occluded, unlike the Mobileye. In contrast to the Mobileye, the Velodyne detects oncoming, passing, crossing and trailing traffic. It can also vehicles traveling in the same direction ahead of the vehicle which are not fully visible to the Mobileye. However, the Velodyne's obstacle detection range in front of the vehicle is much shorter, and the Mobileye can more reliably detect vehicles which are far away. Therefore, the vast majority of the obstacles detected by both sensors do not match. Typically, only 1 or 2 obstacles detected by the both sensors correspond to the same obstacle.

4.5.1 Coordinate Transformation

Both the Mobileye and the Velodyne provide the data in their local reference frames. The obstacle locations therefore must be converted to a common reference frame. During association the position of the Mobileye Obstacle is transformed into the Velodyne frame using the ROS Transform Listener [14].

4.5.2 Mobileye Tracking

The sensor fusion node keeps track of the current Mobileye obstacles. Mobileye Obstacles frequently disappear and reappear between frames, therefore the node removes obstacles from the Mobileye list if it does not receive a new message for a given obstacle within a timeout threshold.

4.5.3 Mobileye and Velodyne Obstacle Association

When the sensor fusion node receives a message from the Velodyne node, it associates the Velodyne and Mobileye obstacles using the Kuhn-Munkres algorithm in the same manner as the data association for the Velodyne tracking node. The cost of an association is the distance between the obstacles at a given time in the Velodyne frame. The distance between the obstacles is then calculated as the distance between the transformed Mobileye position and the interpolated position of the Velodyne obstacle at the time of detection of the Mobileye obstacle. If the distance between two obstacles matched by the Kuhn-Munkres algorithm is above a threshold value, the obstacles are not associated.

4.5.4 Registration

If a Mobileye and Velodyne obstacle are matched, the position, size, relative velocity are set to the position, size, and relative velocity of the Velodyne obstacle, since the Velodyne position and size is much more accurate and reliable, while the type of the obstacle is set to the Mobileye type, either a vehicle, truck, motorcycle, bicycle, or pedestrian.

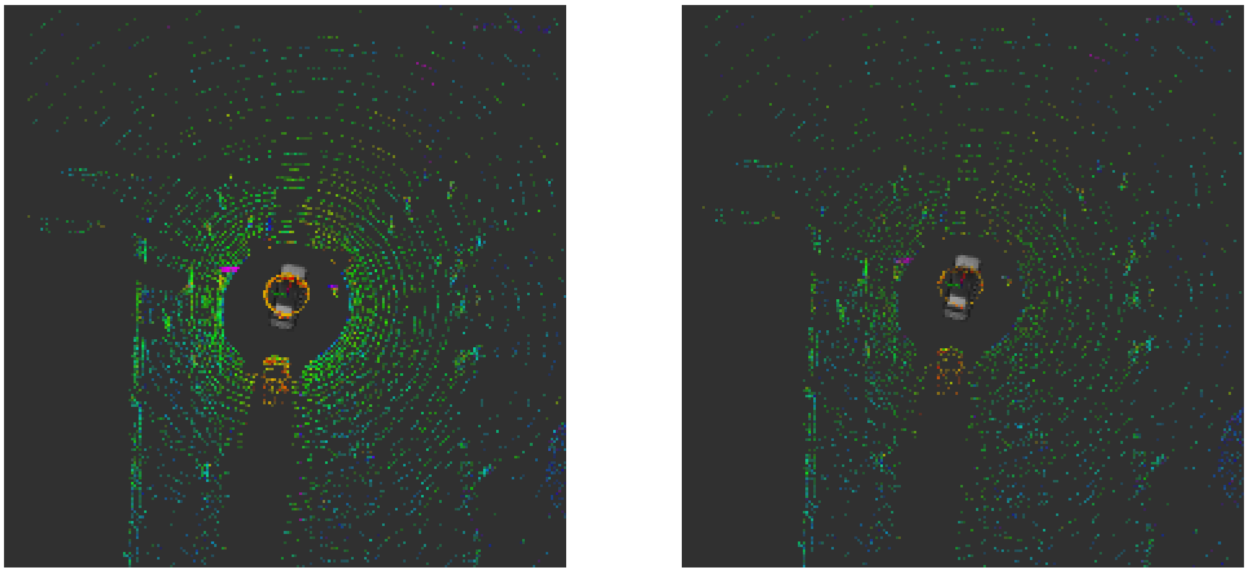
Chapter 5

Results

5.1 Velodyne Feature Extraction

5.1.1 Downsampling

The voxel grid filter used to downsample the cloud uses a leaf size of 0.10 m for the x, y, and z coordinates. The original point cloud is approximately 222,000 points, and after downsampling the point cloud is approximately 55,000 points.



(a) Original point cloud

(b) Downsampled point cloud

Figure 5.1.1: Point cloud at an intersection before and after downsampling

5.1.2 Ground plane estimation

The ground plane estimator was tested with both an unconstrained plane model and a plane model with the normal vector constrained to be within a threshold angle of the normal vector of the ground plane from the previous frame. Threshold angles between 1° and 5° were tested. The performance of the ground plane estimation module was compared in several different environments, including while driving along a bridge, while the vehicle was stopped at a busy intersection, during a sharp left turn, and while turning and driving up. For all of the tests, the inlier distance threshold was set at 0.20 m and the maximum number of iterations was set to 500. The number of points matching the ground plane model was used as the comparison metric since the ground plane estimator slightly under-segments the ground plane.

Table 5.1.1 shows the results of the ground plane segmentation test along a straight flat road with no crossing traffic, as shown in Figure 5.1.2. The aggregate results for each model are shown for over 5 test runs. The constrained ground plane with a tolerance of 5° performs the same as the unconstrained ground plane. In this case, the high threshold is effectively the same as an unconstrained plane. The constrained plane with a tolerance of 1° best segments the ground plane overall. It has the highest mean number of inliers and lowest standard deviation. However, the minimum number of inliers is 40% lower than the minimum number of inliers for the unconstrained ground plane model, indicating that for some clouds, the constrained model poorly segments the ground plane even along flat terrain.

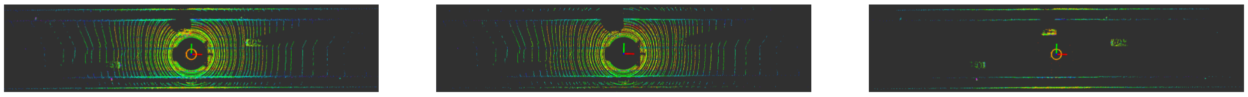

(a) Original point cloud (b) Segmented ground point cloud (c) Above ground point cloud

Figure 5.1.2: Ground estimation along a bridge

	Mean	Stdev	Min	max
Constrained $\epsilon = 1°$	21,011	1,716	7,570	23,203
Constrained $\epsilon = 2.5°$	20,663	1,997	7,819	23,342
Constrained $\epsilon = 5°$	20,606	1,992	12,865	23,342
Unconstrained	20,606	1,992	12,865	23,342

Table 5.1.1: Comparison of ground plane extraction models on straight flat terrain. The comparison metric is the number of inliers for the plane model. These values represent the aggregate mean and standard deviation for 5 test runs. The minimums and maximums were the same for each test run. Each test was run for 30 seconds and the data represents the mean and standard deviation of the number of inliers over the duration of the test.

Table 5.1.2 shows the results of the ground plane estimation test while the vehicle was stopped at a busy intersection shown in Figure 5.1.3. The test was run for 50 seconds. The constrained plane model with a threshold of $\epsilon = 1°$ performed better than the other models. The aggregate mean number of inliers was slightly higher and the aggregate standard deviation was lower. Furthermore, the minimum number of inliers for the 1° model was 12,943 which was higher than the unconstrained model (10,172) and the 2.5° model (12,630). Even the worst performance of the 1° constrained model was better than the other plane models.

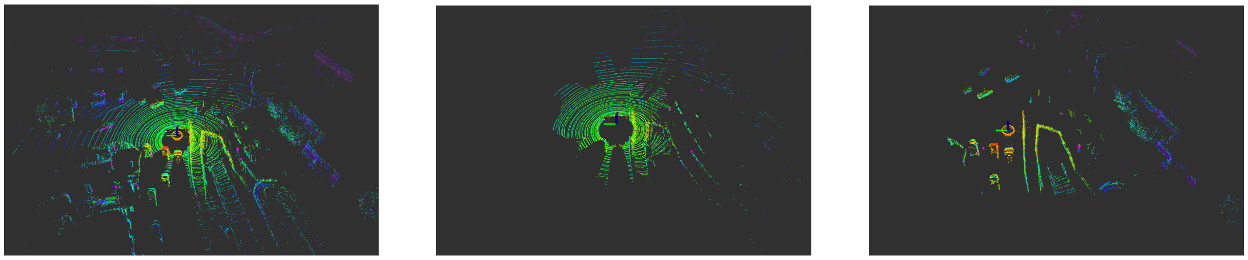

(a) Original point cloud (b) Segmented ground point cloud (c) Above ground point cloud

Figure 5.1.3: Ground estimation at a busy intersection

	Unconstrained		Constrained					
			$\epsilon = 1°$		$\epsilon = 2.5°$		$\epsilon = 5°$	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ
1	21348.39	1960.83	21832.25	1459.55	21391.15	1866.79	21348.44	2036.50
2	21346.93	1966.14	21835.15	1504.20	21393.85	1865.99	21348.51	1961.56
3	21350.31	1967.97	21833.08	1460.47	21386.81	1866.67	21346.22	1962.85
4	21350.97	1963.04	21830.70	1460.60	21395.29	1861.90	21345.73	1967.67
5	21357.01	1960.15	21832.25	1459.55	21395.29	1861.90	21353.04	1964.19
Aggregate	21350.72	1963.63	21832.68	1468.88	21392.49	1864.64	21348.38	1977.35

Table 5.1.2: Comparison of the number of inliers points in the ground plane for constrained and unconstrained plane models at a busy intersection. The duration of each test run was 50 seconds. The sampling algorithms are non-deterministic, therefore the results varied slightly for each test.

The constrained plane model with a 1° threshold performed the best for tests while crossing an intersection and making a sharp turn. However, in one environment when there were two sharp changes in incline, the unconstrained plane model was slightly better. However, all of the models significantly undersegmented the ground region by about 20% due to the sharp change in slope.

5.1.3 Clustering

The cluster extraction algorithm was tested on a variety of scenes. The minimum cluster size threshold was 50 points and the maximum was 1500 points. A maximum cluster size of under 1500 points caused clusters from target vehicles which were very close to ego vehicle to be rejected. In this case target vehicles would briefly disappear as they passed the ego vehicle. Rejecting clusters from static obstacles based on shape was found to be more effective. The point distance threshold used for clustering was 0.50 m. Figure 5.1.4 shows the results of the cluster extraction along a bridge. The three vehicles are all represented by distinct clusters and the guardrails along the bridge are also separated into 6 different clusters. In some cases the clustering algorithm under segmented the cloud, and grouped in pedestrians with the guardrails between the car and the sidewalk.

Figure 5.1.5 shows the results of the cluster extraction at an intersection. This scene is much busier than the bridge scene, there are six vehicles in the scene (shown in colors ranging from red to yellow), all of which are separated into different clusters. The algorithm also clustered 2 trees as distinct clusters as well as features from buildings and the road.

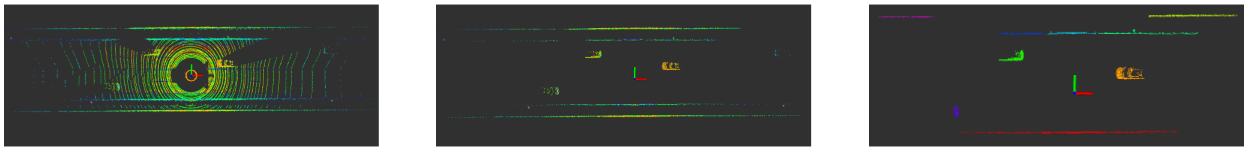

(a) Original point cloud (b) Above ground point cloud (c) Clustered point cloud

Figure 5.1.4: Cluster extraction in a bridge scene. 9 clusters are identified, each is marked with a different color.

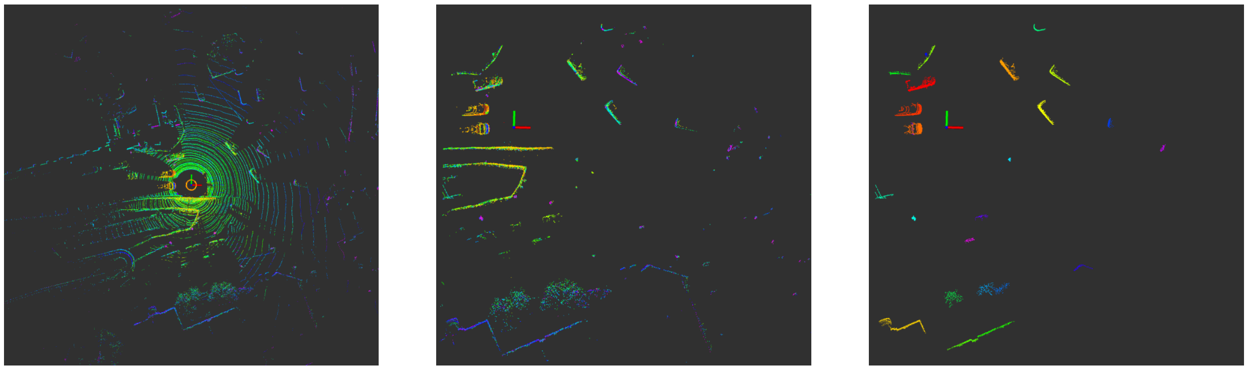
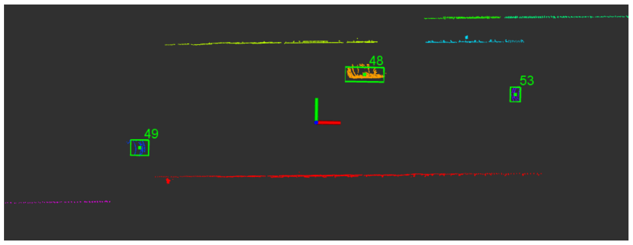

(a) Original point cloud (b) Above ground point cloud (c) Clustered point cloud

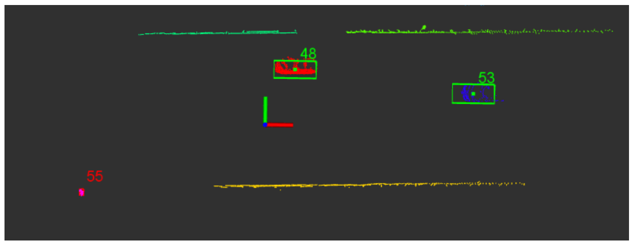
Figure 5.1.5: Cluster extraction at an intersection scene. 25 clusters are identified, each is marked with a different color.

5.2 Velodyne Tracking

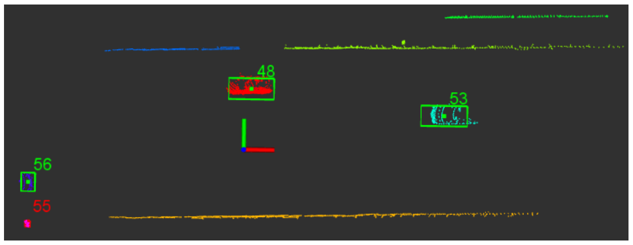
Figure 5.2.1 shows the results of the tracking algorithm while the vehicle drives across a bridge. The algorithm is able to track cars as boxes and pedestrians as points across multiple frames as both the vehicle and obstacles move. Figures 5.2.1a to 5.2.1f show how the algorithm tracks 2 cars with the box model across several frames until the obstacles disappear out of frame. It is able to consistently label the cars with the same ID numbers across several frames and recycles the ID numbers when the cars disappear from view. As shown in Figures 5.2.1c to 5.2.1h, the algorithm is able to track obstacle 56, which is single trailing vehicle traveling at approximately the same velocity for about 10 seconds. They also show that the tracking algorithm is able to detect when obstacles disappear from view. In this particular example, only figure 5.2.1g shows an obstacle modeled as a short line segment, in this case the obstacle is a false positive from ground points which were not removed. The clusters from the guardrails of the bridge are not tracked because they are rejected by the tracking algorithm since they are long line segments.



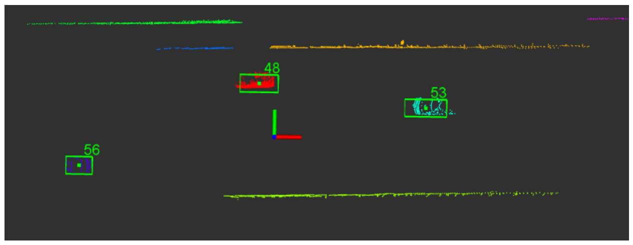
(a) Frame 1



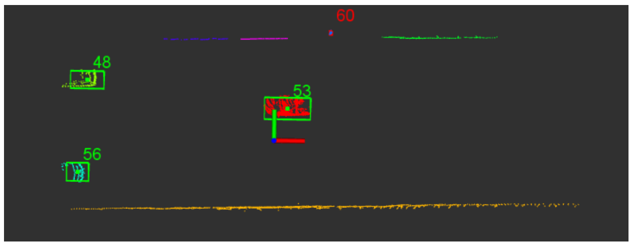
(b) Frame 2



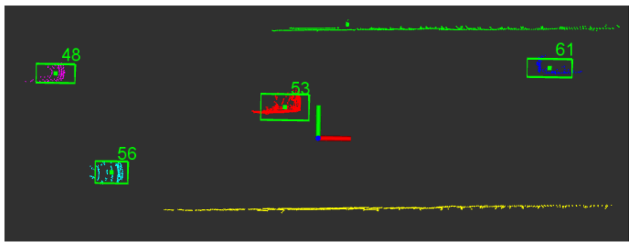
(c) Frame 3



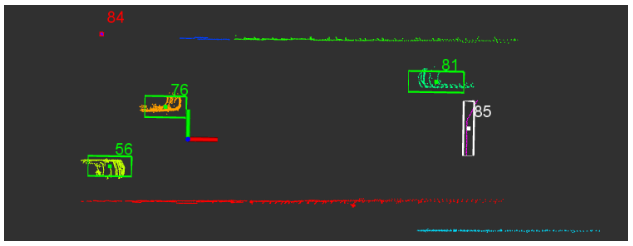
(d) Frame 4



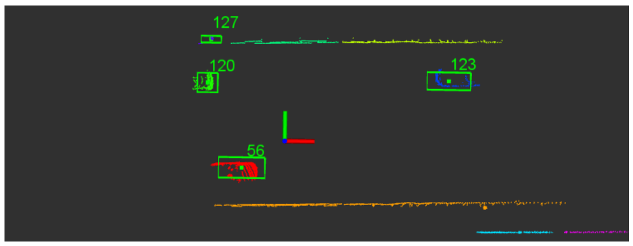
(e) Several frames later, each of the obstacles in frame 4 are still tracked



(f) One frame later, a new obstacle is present



(g) About 5 seconds later, obstacle 56 is still tracked



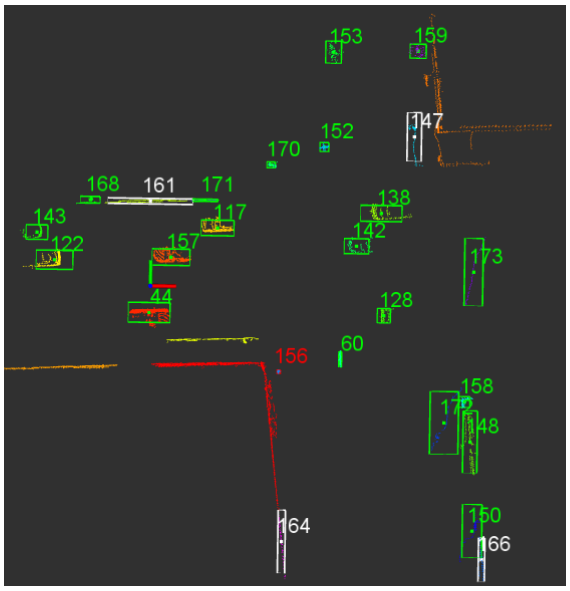
(h) Again, about 5 seconds later, obstacle 56 is still tracked.

Figure 5.2.1: Tracking Velodyne clusters in a bridge scene. The visualization of the clusters does not color a given cluster the same each frame. Tracked obstacles are surrounded by bounding boxes and marked with the tracking ID number. Obstacles which are modeled as points have red labels, those modeled as boxes have green labels, and those modeled as lines have white labels.

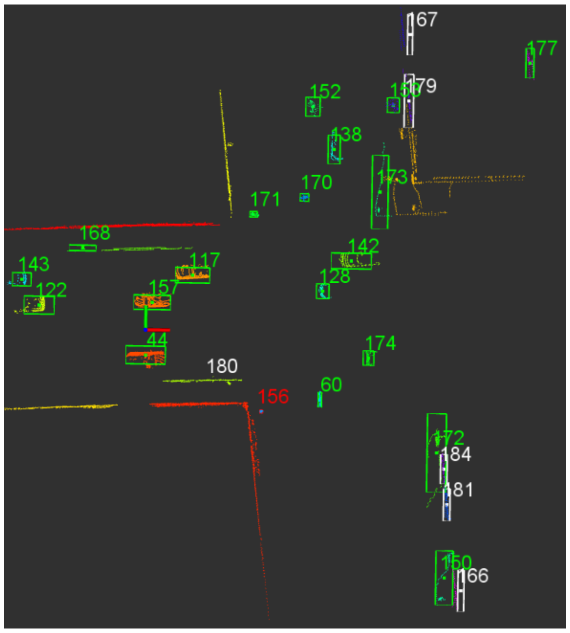
Figure 5.2.2 shows the results of the tracking algorithm as the vehicle approaches an intersection. Several obstacles switch IDs with nearby obstacles between frames. Between the first frame shown in 5.2.2a the second frame shown in 5.2.2a, 5 vehicles switch ID numbers. With the Munkres algorithm, if one obstacle is matched incorrectly, this can cause a cascading effect, such that several obstacles switch ID numbers.

In the third frame a new point cluster is detected and labeled with ID 189, but it is not detected again in the fourth frame, however, a different point obstacle first in frame 4 across the intersection and is assigned label 189 because it is the closest match. In this case, since the original point disappears and a new one appears, it is matched to the new obstacle because it is the closest one.

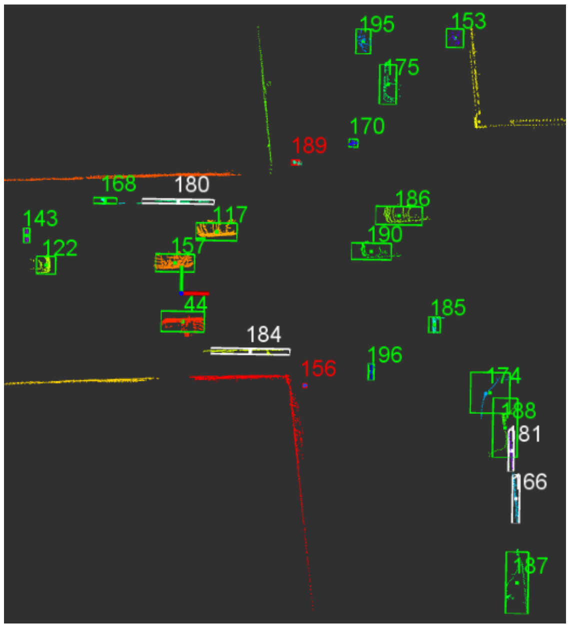
Obstacles 44, 122, 143, and 156 remain stable across all of the frames since they do not move significantly relative to the vehicle between frames. While the tracking algorithm fails to label obstacles correctly in this scene, it does not lose track of any vehicles or pedestrians, but does switch labels. The label switching occurs in part because the relative velocity is assumed to be constant between frames, but in this case, the vehicle is decelerating as it approaches the intersection. Therefore, the interpolated position of the obstacles from the previous used for matching will have higher error. Taking the vehicle's velocity and acceleration into account would result in more accurate matching.



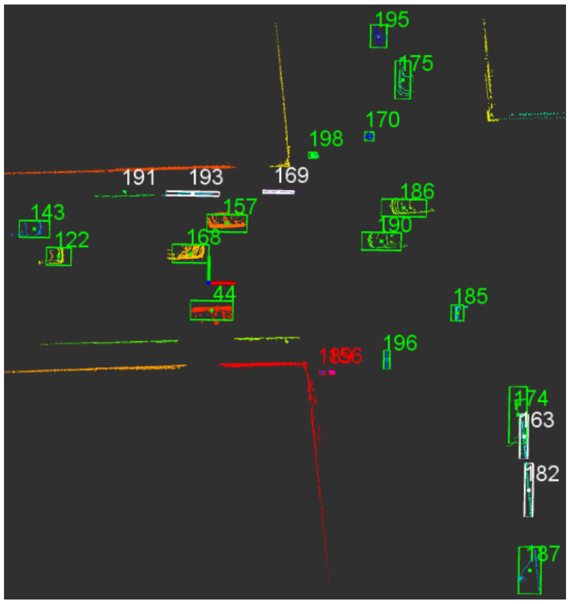
(a) Frame 1



(b) Frame 2, Vehicles with IDs 128, 142, 138, 153, 174, and 138 all switch ID numbers



(c) Frame 3



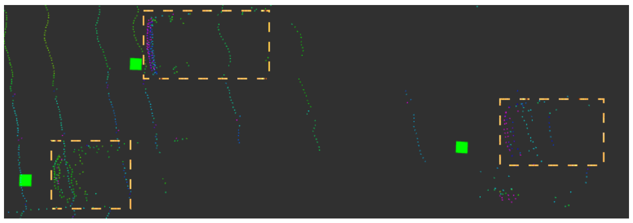
(d) Frame 4

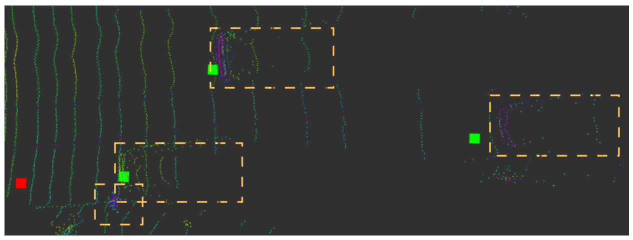
Figure 5.2.2: Tracking obstacles while approaching an intersection, the visualization of the clusters does not color a given cluster the same each frame. Tracked obstacles are surrounded by bounding boxes and marked with the tracking ID number. Obstacles which are modeled as points have red labels, those modeled as boxes have green labels, and those modeled as lines have white labels.

5.3 Mobileye to Velodyne Coordinate Transform

As mentioned in previous sections, the transform between the Mobileye and Velodyne was estimated by measuring the distance between the Mobileye and a common reference frame. This technique proved to be robust enough to accurately associate Mobileye and Velodyne obstacles. As shown in 5.3.1, the transformed Mobileye coordinates are very close (within 50 cm) to the closest sides of obstacles facing the front of the vehicle in the Velodyne coordinate frame and thus can be correctly matched to the nearest Velodyne obstacle. Across several frames, the error between transformed Mobileye positions and the location of the obstacles in the Velodyne frame is not consistent. This is partly due to timing differences and the motion of the vehicle and targets. The error in the Mobileye position estimation for pedestrians is typically larger than the error in the estimated position of vehicles, and in this example shown in the figure the error in the estimation of the pedestrian location is between 2 and 4 meters in both frames.

Errors in transformation are caused by calibration error, timing error, and the vehicle and obstacle motion. Since the Mobileye does not provide intrinsic calibration information, it is not possible to isolate the source of the error. However, by simply transforming using the extrinsic measurements, the position errors are generally small enough (within 1 meter) that the fusion layer can correctly match obstacles in a single frame.


(a) Frame 1. Three vehicles are identified.


(b) Frame 2. Three vehicles and one pedestrian are identified.

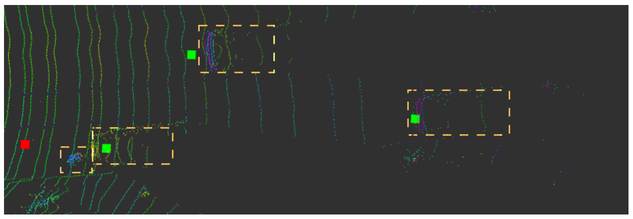
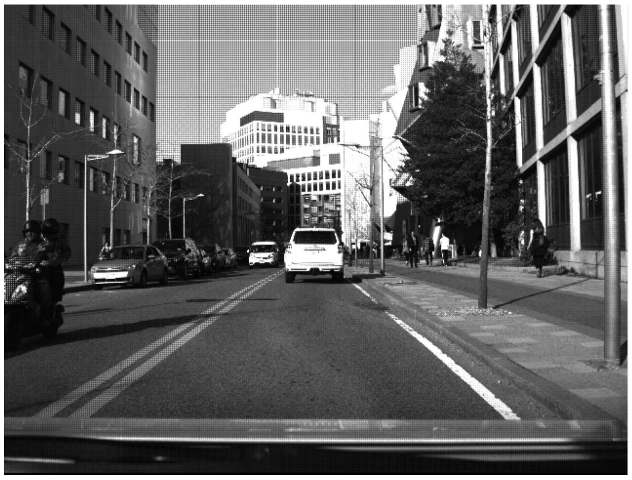

(c) Frame 3. Three vehicles and one pedestrian are identified.

Figure 5.3.1: An example showing the position of the Mobileye obstacles shown as dots transformed into the Velodyne frame and overlayed onto the point cloud. Green dots represent vehicles and red represent pedestrians. The bounding boxes around the Velodyne obstacles identified by the Mobileye are hand drawn for this example.

5.4 Sensor Fusion

The Sensor Fusion Module associated Mobileye and Velodyne obstacles in busy scenes fairly robustly. However, in urban streets, the vehicle frequently accelerated or decelerated and therefore the Velodyne tracking was not very robust. While the Velodyne module was able to reliably detect obstacles within it's field of view, the labels for obstacles frequently changed. Although the labeling of the Velodyne obstacles was unreliable, the Sensor Fusion Module was still able to reliably match the Velodyne and Mobileye obstacles each frame. Figure 5.4.2 shows the results of the fusion on a busy urban street shown in 5.4.1. The Mobileye reliably tracks the closest in path vehicle, labeled M51. On the other hand, the Velodyne detects the vehicle in each frame, it does not track it reliably across frames and switches its label. The Velodyne Tracker assumes the velocity is constant between frames, and in this case the vehicle is decelerating, which results in an inaccurate estimation of the target vehicle's updated position. In spite of the Velodyne module's inaccurate labeling, the sensor fusion module matches the correct Velodyne obstacle to the detected Mobileye vehicle each frame. As shown in each of the figures, the Mobileye also detects several pedestrians. The original point cloud shows these pedestrians, but each of the clusters contains too few points, and are partially obscured by an overhanging tree.


(a) The closest in path vehicle and pedestrians are clearly visible

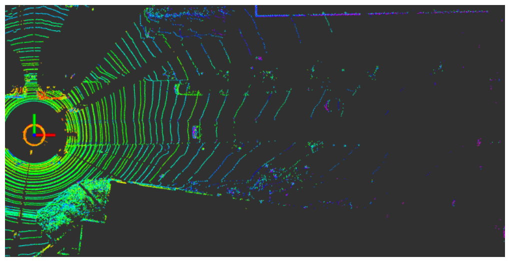
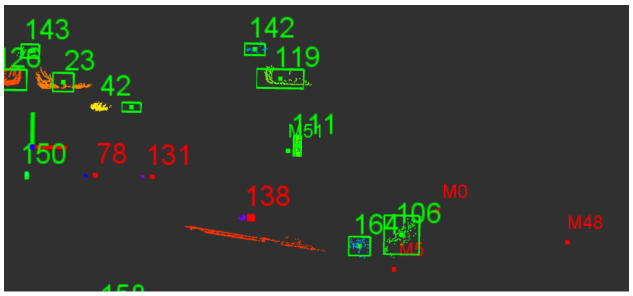
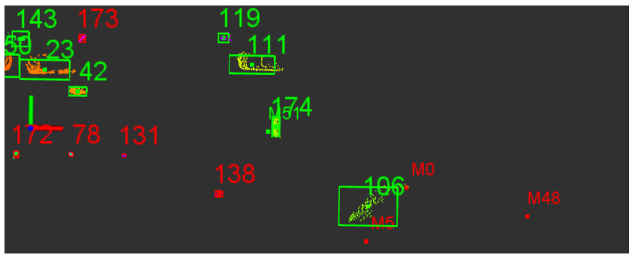

(b) The point cloud (partially cropped) corresponding to the first frame of the scene. The closest in path vehicle is visible, as are the pedestrians detected by the Mobileye, but they are obscured by an overhanging tree and walking too close together for the feature extraction module to detect them properly.

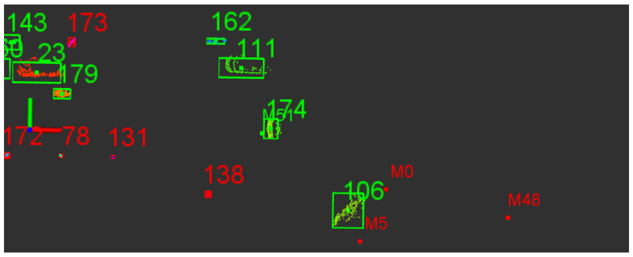
Figure 5.4.1: To provide context for the results shown in Figure 5.4.2, this figure shows an image from the Point Grey camera and the point cloud corresponding the first frame of the scene.



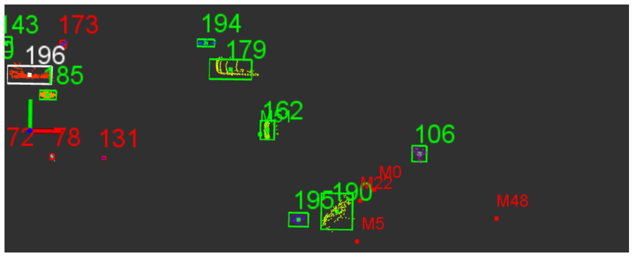
(a) Frame 1, Matches (M51, 111)



(b) Frame 2, Matches (M51, 174))



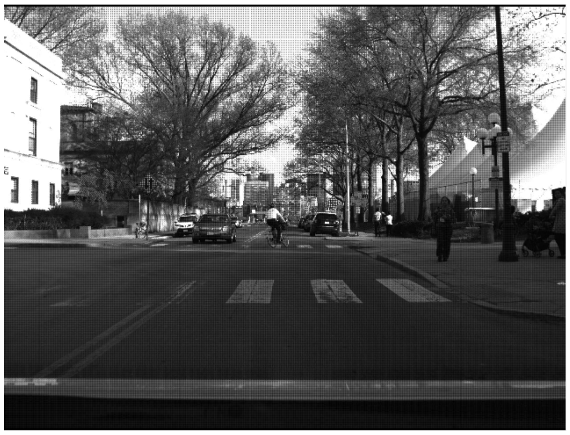
(c) Frame 3, Matches (M51, 174)



(d) Frame 4, Matches (M51, 162)

Figure 5.4.2: Fusion in an urban street scene. The visualization of the clusters does not color a given cluster the same each frame. Tracked Velodyne obstacles are surrounded by bounding boxes and marked with the tracking ID number. Obstacles which are modeled as points have red labels, those modeled as boxes have green labels, and those modeled as lines have white labels. ID's of tracked Mobileye obstacles are labeled with M. For Mobileye obstacles, pedestrians are red and vehicles are green.

In a slightly less busy urban scene shown in Figure 5.4.3, the Sensor fusion module is able to track a few obstacles across several frames, as shown in Figure 5.4.4. In each frame, the Mobileye detects a parked vehicle, labeled M25, on the side of the road. This vehicle is not detected by the Velodyne in any of the frames because the target returns too few points. The sensor fusion module instead matches the vehicle with a nearby Velodyne feature along the side of the road which is not a vehicle. Furthermore, the Mobileye identifies an oncoming vehicle, labeled M62, which is matched to the correct Velodyne target in each frame, although the Velodyne label changes in the third frame. The Mobileye also identifies a pedestrian, labeled M42, on the sidewalk. In each frame, the Velodyne identifies both the pedestrian and a light pole as point features near the Mobileye target. In the first and last frame, the obstacle is correctly matched, but in the other two frames it is matched with the light pole. Finally, the Mobileye detects a bicycle, labeled M59, but classifies it as a pedestrian. The sensor fusion module attempts to match the Mobileye target with Velodyne obstacles modeled as points, and finds no match.


(a) A pedestrian is walking along the sidewalk to the right of the vehicle near a light pole, a bicycle is in front of the vehicle turning into the vehicle's lane, an oncoming car is in the next lane, and parked vehicles are visible along the right side.

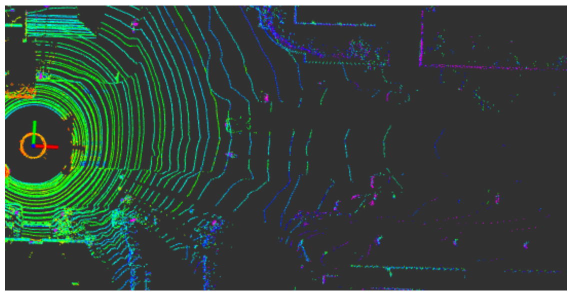
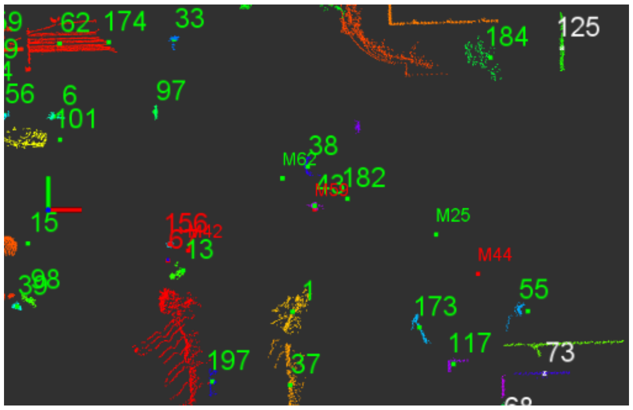
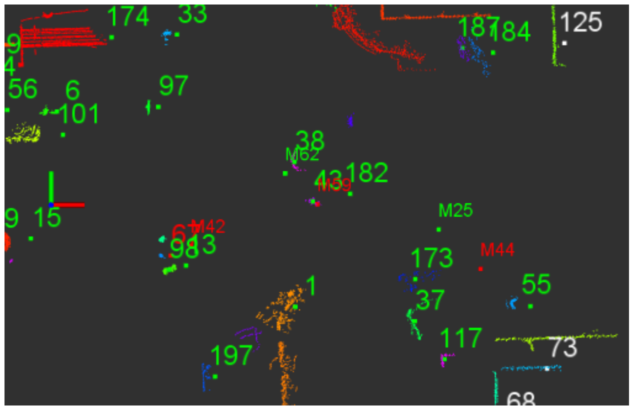

(b) The bicycle and oncoming vehicle are visible, but the parked vehicle is not.

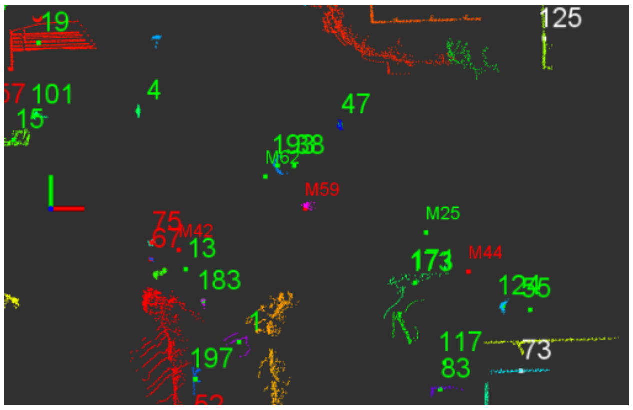
Figure 5.4.3: To provide context for the results shown in Figure 5.4.4, this figure shows an image from the Point Grey camera and the point cloud corresponding the first frame of the scene.



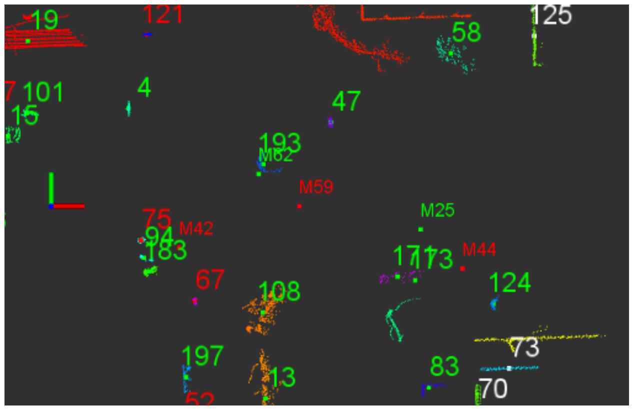
(a) Frame 1. Matches (M25, 173), (M42, 156), (M62, 38).



(b) Frame 2. Matches (M25, 173), (M42, 67), (M62, 38)



(c) Frame 3. Matches (M25, 171), (M42, 67), (M62, 193)



(d) Frame 4. Matches (M25, 171), (M42, 75), (M62, 193).

Figure 5.4.4: Fusion in an urban street scene. The visualization of the clusters does not color a given cluster the same each frame. Tracked Velodyne obstacles are surrounded by bounding boxes and marked with the tracking ID number. Obstacles which are modeled as points have red labels, those modeled as boxes have green labels, and those modeled as lines have white labels. ID's of tracked Mobileye obstacles are labeled with M. For Mobileye obstacles, pedestrians are red, bicycles are magenta, and vehicles are green.

When approaching a busy intersection shown in Figure 5.4.5, the Sensor fusion module is able to track the leading and neighboring vehicle for several seconds, as shown in Figure 5.4.6. The Mobileye detects the closest in path vehicle, labeled M37, and a vehicles in the next lane, with labels M35 and M54, the Velodyne tracker also detects these vehicles each frame and labels them consistently for several seconds. Each frame the sensor fusion module correctly matches Mobileye obstacle 37 with Velodyne obstacle 44 and Mobileye obstacle 35 with Velodyne obstacle 17. Furthermore, the Mobileye obstacle M54 is only detected by the Velodyne in Figure 5.4.6d and it is associated correctly in this frame. In the other frames, the target is incorrectly matched to the vehicle to the right of the Mobileye obstacle M54.

The Mobileye also detects parked vehicles, M29 and M31. The Velodyne Module initially does not detect the obstacle corresponding to M31, and the Sensor Fusion Module incorrectly matches it to a nearby obstacle. In the second frame, the Velodyne Module detects it and the sensor fusion module is able to track it for 1 second. Between the frames shown in Figure 5.4.6c and Figure 5.4.6d, the Velodyne Module changes the label of the parked vehicle corresponding to M31, but the Sensor Fusion Module continues to correctly match Mobileye Obstacle 31 to the new Velodyne label. The Velodyne Module only detects an obstacle corresponding to M29 in the final frame and it is correctly matched.

(a) Both closest in path vehicle and leading next lane vehicle are clearly visible, as well as a parked vehicle along the right.

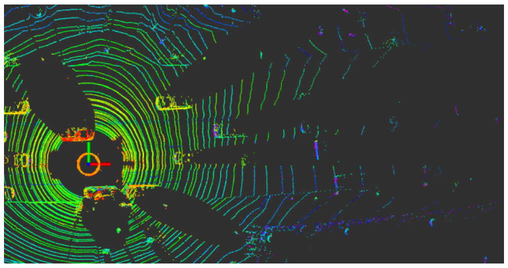
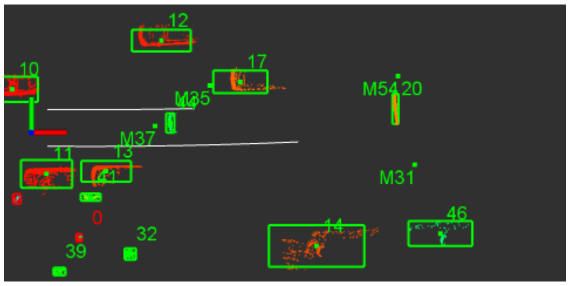
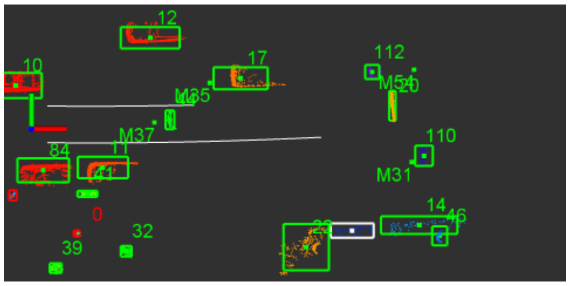

(b) The closest in path vehicle and leading next lane vehicle is visible. Oncoming vehicles are also visible. But the parked vehicle only returns a few points.

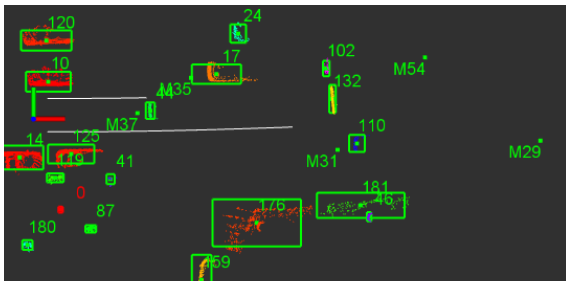
Figure 5.4.5: To provide context for the results shown in Figure 5.4.6, this figure shows an image from the Point Grey camera and the point cloud corresponding the first frame of the scene shown in 5.4.6a.



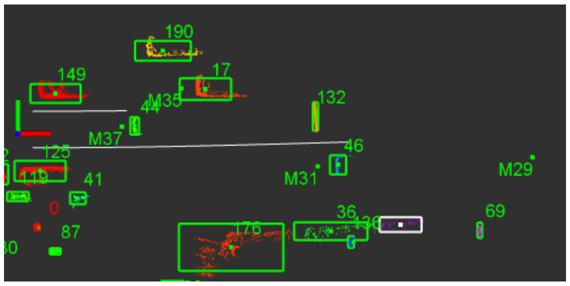
(a) Frame 1. Matches (M31, 46), (M35, 17), (M37, 44), (M54, 20)



(b) Next frame. Matches (M31, 110), (M35, 17), (M37, 44), (M54, 112)



(c) 1 second later. Matches (M31, 110), (M35, 17), (M37, 44), (M54, 102)

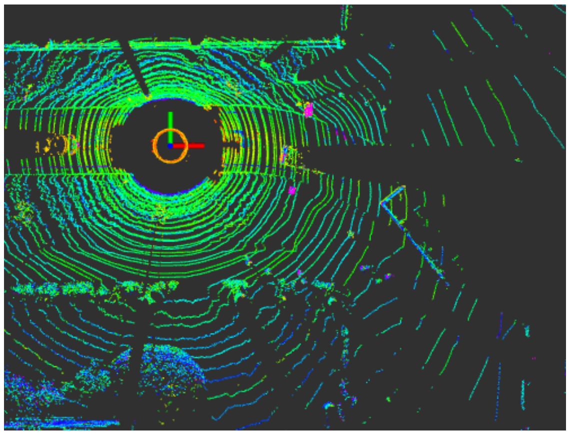


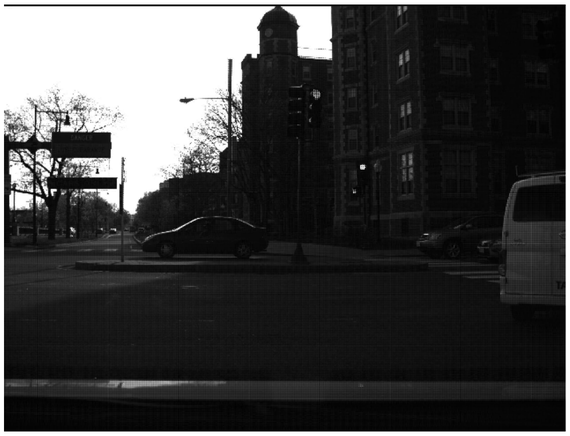
(d) 1 second later. Matches (M29, 69), (M31, 46), (M35, 17), (M37, 44)

Figure 5.4.6: Fusion at an intersection. The visualization of the clusters does not color a given cluster the same each frame. Tracked Velodyne obstacles are surrounded by bounding boxes and marked with the tracking ID number. Velodyne obstacles which are modeled as points have red labels, those modeled as boxes have green labels, and those modeled as lines have white labels. Mobileye Obstacle ID's are indicated with M. Mobileye pedestrians are red, bicycles are magenta, and vehicles are green.

The example shown in Figure 5.4.8 shows the results of the Sensor Fusion while the vehicle is turning in a busy intersection shown in 5.4.7. As the lead vehicle turns, the Mobileye loses track of the target, but the Velodyne module detects it in each frame, although it the label changes twice. The Sensor Fusion Module correctly matches the target when both sensors detect it. In this scene, the Mobileye is only able to detect the leading vehicle and no other obstacles, but the Velodyne is able to detect the leading vehicle and vehicles ahead of it, as well as crossing traffic, oncoming traffic and pedestrians.

(a) The Point Grey image corresponding to Figure 5.4.8a.


(b) The Point cloud corresponding to Figure 5.4.8a.


(c) The Point Grey image when the Mobileye loses track of the vehicle, one frame after Figure 5.4.8c.

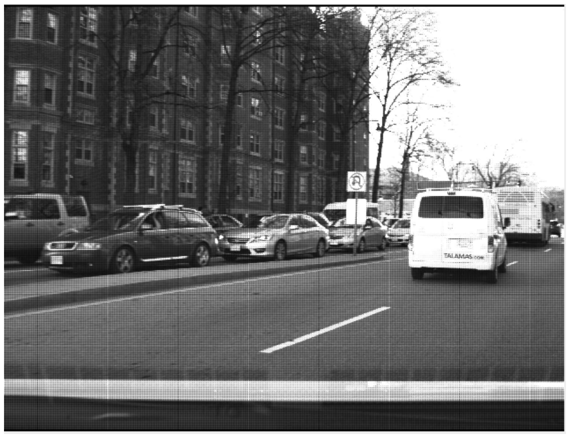
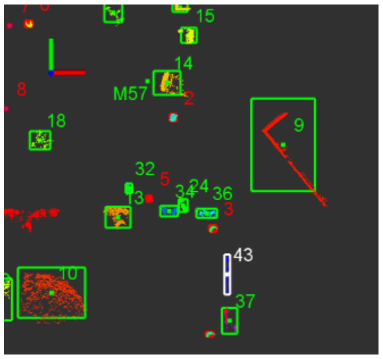
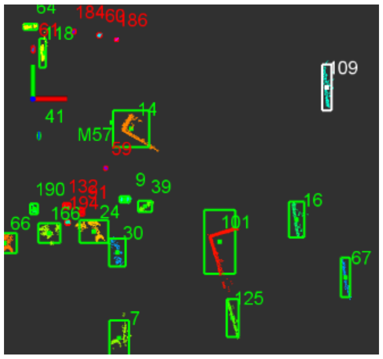

(d) The Point Grey image when the Mobileye detects the vehicle, corresponding to Figure 5.4.8e.

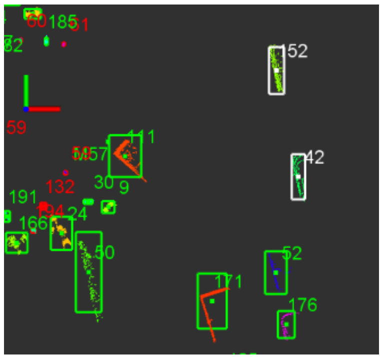
Figure 5.4.7: To provide context for the results shown in Figure 5.4.8, this figure shows an image from the Point Grey camera and the point cloud corresponding the first frame of the scene shown in 5.4.8a.



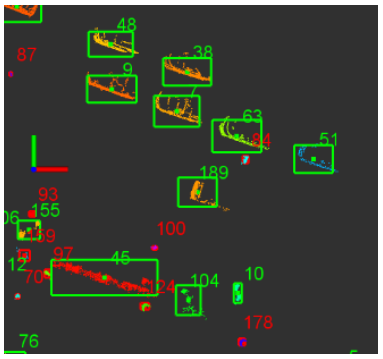
(a) Frame 1, before the turn, the closest in path vehicle is turning. The Mobileye and Velodyne both detect the vehicle, and the targets are correctly matched.



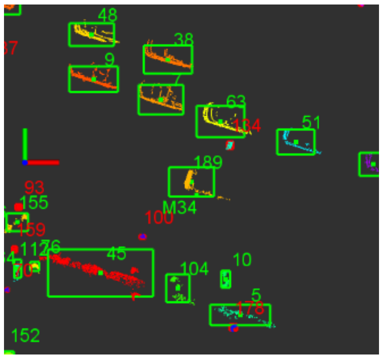
(b) Approximately 2 seconds later, the target is labeled consistently and correctly matched.



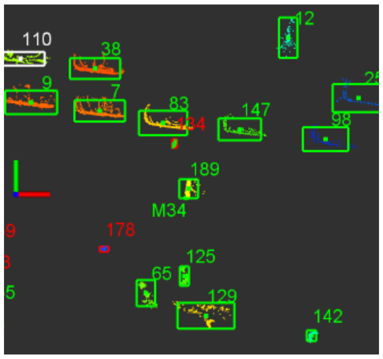
(c) Approximately 1 second later, the Velodyne label changes, but the target is matched correctly. The Mobileye stops detecting the obstacle in the next frame.



(d) Approximately 1.5 seconds later, the Velodyne label changes again and the Mobileye still does not see lead vehicle.



(e) The next frame (approx. 0.10 seconds later), the Mobileye again detects the lead vehicle, and the targets are correctly matched.



(f) Approximately 0.5 seconds later, the turn is complete. The Velodyne labels persist and the targets are correctly matched.

Figure 5.4.8: Fusion while the vehicle is turning. The visualization of the clusters does not color a given cluster the same each frame. Tracked Velodyne obstacles are surrounded by bounding boxes and marked with the tracking ID number. Velodyne obstacles which are modeled as points have red labels, those modeled as boxes have green labels, and those modeled as lines have white labels. Mobileye Obstacle ID's are indicated with M. Mobileye pedestrians are red, bicycles are magenta, and vehicles are green.

Chapter 6

Conclusion

The Mobileye is primarily useful for detecting lanes and the closest in path vehicle, but can not be relied upon for more sophisticated target tracking. For navigation in urban environments, the Velodyne sensor detects obstacle much more reliably. Furthermore, the Mobileye performance does not exactly match the documentation provided by AutonomousStuff [15], for example, as mentioned previously, the reference point message always reported a value of 0.0, 0.0, when the documentation stated that the reference point should be located at a 1 second headway ahead of the vehicle. Since Mobileye does not provide confidence values on the position, relative velocity, or types of obstacles, it is difficult to apply reliable filtering to the tracked Mobileye objects. It is also difficult to calibrate the Mobileye with other sensors since the intrinsic calibration measurements are not provided. The Mobileye would be a more useful sensor if these values were provided.
For feature extraction from the point cloud, estimating the ground as a plane and Euclidean Cluster Extraction provided reliable segmentation in urban environments. The Velodyne tracker node was able to reliably detect obstacles within the Velodyne's field of view, but the tracking was only robust in simple scenes like bridges. The tracker can be improved in several ways, first by incorporating the road geometry to eliminate off road obstacles. Furthermore, a better motion estimation model, such as the Constant Turn Rate Acceleration Model, combined with either Unscented Kalman Filtering or an Extended Kalman Filter would improve target tracking. Using Multi Hypothesis Tracking would also improve the trackers capabilities.
The Mobileye and Velodyne are primarily complimentary sensors. Most of the obstacles that the Velodyne detects are not detectable by the Mobileye and vice versa. The Mobileye tracks the closest in path vehicle robustly, but is not as reliable for tracking other obstacles in urban environments because of occlusions. The Velodyne tracker works well in environments like bridges, where it does not detect as many obstacles and can track obstacles more easily. Although the Velodyne tracker frequently incorrectly associated obstacles

between frames in busy scenes, it still was able to reliably detect the visible obstacles in the scene. The Sensor Fusion module was therefore able to detect obstacles visible to either sensor in each scene and track them reliably. The target Velodyne obstacle which was associated with a Mobileye Obstacle remained the same between frames although the Velodyne labels sometimes changed. This demonstrates that with improved Velodyne tracking, the Sensor Fusion module can provide robust tracking of fused Mobileye and Velodyne obstacles.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my thesis advisor Professor Sertac Karaman for the continuous support of my research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I would also like to thank the Toyota Research Institute for the opportunity to join the project and Professor Daniela Rus and Dr. Liam Paull for their insightful comments and guidance. Finally, I am grateful to my fellow researchers on the project, in particular Teddy Ort, who helped me collect data on several occasions, and Stephen Proulx, for working on the hardware integration and calibration.

Bibliography

[1] Michael S. Darms, Paul E. Rybski, Christopher Baker, and Chris Urmson. Obstacle detection and tracking for the urban challenge. In *IEEE Transactions on Intelligent Transportation Systems, IEEE*, 2009.

[2] M. Himmelsbach, T. Luettel, and H.-J. Wuensche. Real-time object classification in 3d point clouds using point feature histograms. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[3] P. Morton, B. Douillard, and J. Underwood. An evaluation of dynamic object tracking with 3d lidar. In *Australasian Conference on Robotics and Automation*, Dec 2011.

[4] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel. On the segmentation of 3d lidar point clouds. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[5] F. Moosmann, O. Pink, and C. Stiller. Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion. 2009.

[6] James Munkres. Algorithms for the assignment and transportation problems. In *Journal of the Society for Industrial and Applied Mathematics*, 1957.

[7] R Omar Chavez-Garcia and Olivier Aycard. Multiple sensor fusion and classification for moving object detection and tracking. In *IEEE Transactions on Intelligent Transportation Systems, IEEE*, 2015.

[8] Liang Zhang, Qingquan Li, Ming Li, Qinqzhou Mao, and Andreas Nuchter. Multiple vehicle-like target tracking based on the velodyne lidar. In *IFAC*, 2013.

[9] Faraz M Mirzaei, Dimitrios G Kottas, and Stergios I Roumeliotis. 3d lidarâĂŞcamera intrinsic and extrinsic calibration: Identifiability and analytical least-squares-based initialization. In *The International Journal of Robotics Research*, 2012.

[10] Jack O'Quin. Ros: Velodyne package summary.

[11] Velodyne hdl-64e.

[12] Mobileye accident prevention system advanced driver assistance system.

[13] Mobileye 630 pro mit vibrationsalarm - incl. montage vor ort - bundesweit.

[14] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, , and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[15] AutonomousStuff. Mobileye startup guide v 1.7, 2016.

[16] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

Appendices

Appendix A

Mobileye ROS Messages

A.1 Obstacle Messages

A.1.1 MobileyeObstacleStatus

CAN Message ID: 0x738 This message contains information about all of the obstacles in the scene including the number of obstacles and whether a car is close in front of the vehicle. It also contains the stop/go recommendation if the vehicle is stopped and the failsafe mode being used.

A.1.2 MobileyeObstacle

CAN Message IDs: 0x739 - 0x765 This message describes a dynamic obstacle in the scene. The types of dynamic obstacles detected include vehicles, trucks, motorcycles, bicycles, and pedestrians. The message includes the obstacle's ID number, type, position relative to the reference point, relative x (forward) velocity, acceleration, width, length (if the front of the vehicle is viewable), and lane position. If the obstacle is in the ego lane or the next lane, it describes whether or not the vehicle is cutting in or out of the ego lane. Furthermore it includes the angle to the center of the obstacle and the scale change from the previous frame. If the obstacle is a vehicle it also includes information about the obstacle's blinker status, brake light status, and whether it is the closest-in-path vehicle (CIPV). Finally, it includes whether or not it was a newly detected obstacle and the age of the obstacle in number of frames.

A.2 Lane Messages

A.2.1 MobileyeLane

CAN Message ID: 0x737 This contains lane information and measurements for the current lane including the lane curvature and heading as well as the yaw and pitch angle of the vehicle derived from the lane measurements. It also indicates if the vehicle is in a construction area and if the lane departure warning is enabled for the left and right lane marks.

A.2.2 MobileyeEgoLane

CAN Message ID: 0x669 This message contains lane information for the left and right lane markings. Including the distance to the lane marking, the marking type, whether or not Lane Departure warnings are available for each marking, and a confidence grade about the lane marking information.

A.2.3 MobileyeLaneMarking

CAN Message IDs: 0x766 - 0x769, 0x76C - 0x77A This message describes a lane marking. It is used for both the next lane A and B messages and the Lane Keeping Assistant (LKA) A and B messages for both left and right lane markings, all of which have the same parameters. The message âĂIJidâĂİ parameter indicates whether the marking is for the current laneâĂŹs markings (left or right) or the next laneâĂŹs marking. For the current lane, isLeft indicates whether the lane marking is on the left or right. For all lane markings, it describes the distance between the camera and the lane mark, the parameters of a cubic equation describing the shape of the lane, the width of the marking, the physical view range of the marking, and the validity of the view range.

A.2.4 MobileyeNextLanes

CAN Message ID: 0x76B This message indicates the number of next lane markers which were reported.

A.3 Traffic Sign Messages

A.3.1 MobileyeTrafficSign

CAN Message ID: 0x720, ..., 0x726 These messages detail the Traffic Sign type and position relative to the camera. The Mobileye only sends a traffic sign message once the sign has exited the frame. Up to 7 signs can be detected per frame.

A.3.2 MobileyeVisionOnlyTSR

CAN Message ID: 0x727 This messages summarizes the types of detected traffic signs in each frame, including any supplementary signs. The traffic signs are shown in the order they are detected.

A.4 Other Messages

A.4.1 MobileyeFixedFOE: Focus of Expansion

CAN Message ID: 0x650 This message describes the fixed focus of expansion yaw and horizon.

A.5 MobileyeAws: Warning System

CAN Message ID: 0x700 This message contains the Warning System and Display messages. It signals forward collision warnings (FCW) for pedestrians and vehicles, lane departure warnings (LDW), and the headway measurement and warning level. The headway is the minimum time between the vehicle and the next vehicle assuming constant speed. It indicates when a sound should be played and the type of sound that should be played. It also indicates if the Lane Detection algorithms are working properly, a failsafe mode is on, and night or dusk mode is on. The failsafe modes are blur image, saturated image, low sun, partial blockage, and partial transparent. The current failsafe mode is indicated in the obstacle status message.

A.5.1 MobileyeAhbc: Automatic High Beam Control

CAN Message ID: 0x728 This message contains the High Beam / Low Beam recommendation. If low beam is recommended, it indicates the reasons for using low beam (e.g. an oncoming vehicle or fog).

A.5.2 MobileyeLights

CAN Message ID: 0x729 This message contains information about the left, right and lower boundaries of the glare free area of the image and the range of the closest object defining the lower boundary. It indicates the position of the boundary and the state of the boundary (e.g. defined by preceding vehicle). It also indicates if there are too many light sources and if the scene is busy.

A.5.3 MobileyeVehicleState

CAN Message ID: 0x760 This message provides information about the status of the vehicles brakes, left and right turn signal, wipers, and high and low beam, if available. It also provides the vehicle speed in km/h if available.

A.5.4 MobileyeRefPoint

CAN Message ID: 0x76A This message describes the reference points 1 and 2. For each point it contains the distance from the camera to the point and the lateral distance between the camera and the reference point. The first reference points is located at the lane center at approximately the 1 second headway.