EVENT-BASED OBSTACLE DETECTION WITH COMMERCIAL LIDAR

by

Chaz Cornwall

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Scott Budge, Ph.D. Major Professor	Reyhan Baktur, Ph.D. Committee Member
Todd Moon, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2022

ABSTRACT

Event-based Obstacle Detection with Commercial LiDAR

by

Chaz Cornwall, Master of Science

Utah State University, 2022

Major Professor: Scott Budge, Ph.D.
Department: Electrical and Computer Engineering

Autonomous ground vehicles are now a reality thanks to advances in processing power and sensing capabilities. However, successfully detecting obstacles at high speeds remains a challenge since sensors must see objects from very far away or the obstacle detection system must find obstacles very quickly. Fast obstacle detection can be reached by using an event-based processing pipeline that is inspired from biology and neuromorphic systems.

A novel obstacle detection modality, the importance map, is shown to dynamically locate obstacles in a scene using the properties of obstacle motion and pixel-wise processing. To ensure this event-based architecture does not throw away essential information for obstacle avoidance maneuvers, these modalities are used as input for a convolutional neural network designed for obstacle avoidance. The importance map does not degrade the network's obstacle avoidance performance, but rather improves the network's results due to the memory contained within the importance map. Despite the improved performance, the importance map contains vehicle measurement dependencies that could make commercial deployment difficult. Vehicle state estimation is implemented inside the importance map to remove the need for external sensors. The result is an efficient obstacle detection system consisting of a single LiDAR (Light Detection and Ranging) sensor and processing unit.

(95 pages)

PUBLIC ABSTRACT

Event-based Obstacle Detection with Commercial LiDAR

Chaz Cornwall

Computerized obstacle detection for moving vehicles is becoming more important as vehicle manufacturers make their systems more autonomous and safe. However, obstacle detection must operate quickly in dynamic environments such as driving at highway speeds. A unique obstacle detection system using 3D changes in the environment is proposed. Furthermore, these 3D changes are shown to contain sufficient information for avoiding obstacles. To make the system easy to integrate onto a vehicle, additional processing is implemented to remove unnecessary dependencies. This system provides a method for obstacle detection that breaks away from typical systems to be more efficient.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

Obstacle detection (OD) in autonomous vehicles and advanced driver assistance systems (ADAS) has become practical due to recent improvements in processing power and sensing capabilities. OD ensures vehicles travel safely through dynamic environments, whether the vehicles are manned or not. Of the top 10 scenarios of light-vehicle crashes, 7 of those scenarios involve the driver's ability to detect obstacles [1]. Furthermore, over 36,000 people in the United States die in motor vehicle traffic crashes each year [2]. An effective OD system can have a drastic positive impact by reducing lives lost and property damage.

Unfortunately, an effective OD system is difficult to create for dynamic and highway-speed environments, which is a common scenario for light vehicles. In a dynamic environment, it often becomes difficult to discern what objects in the scene are important and which are not due to the abundance of potential obstacles. In a highway-speed environment, the speed of recognizing and reacting to an object must be faster than when the vehicle is travelling slowly. Objects in these types of environments often become emergent obstacles, or objects that quickly enter the driver's field-of-view (FOV) and immediately become obstacles. Emergent obstacles are much more likely in these situations because the FOV is constantly changing and objects are frequently obscured. If carefully designed, the same system can identify emergent obstacles in dynamic and highway-speed environments. Reducing the data to only important information creates less data, which reduces the time needed for processing. Some of the best examples of information-efficient and low-latency systems can be found in biology.

The human brain processes emergent obstacles through the visual subcortical pathway of the brain, which triggers obstacle dodging behaviors instead of recognition and classification [3]. The recognition and classification processing path, or the brain's cortical

pathway, takes 4 times longer to process obstacles than the subcortical pathway. Many state-of-the-art OD systems rely on the complex recognition and classification of obstacles into categories such as road, car, pedestrian, etc. [4]. Emergent obstacles require the fast processing of the subcortical pathway, but these OD systems can be seen as artificial cortical pathways. This creates an obvious mismatch. The human nervous system transmits information to the brain by asynchronous "events." These events are voltage potentials that travel along neurons' axons and across synapses to other neurons. In other words, if nothing "important" is occurring, nothing is sent to the brain. Using inspiration from the brain's subcortical pathway and neuron's synapses, possible OD design principles are to use simple classifications and low-level asynchronous "events" for finding obstacles.

The concept of information efficiency is also prevalent in communication system theory. At the beginning of the computer age, Claude Shannon proposed a quantitative measure of information by using the probability of outcomes [5]. If an individual is outside and can see the sun is shining, a passerby informing this individual that the sun is shining provides no information to the individual. If this same individual goes in a building without windows for several hours and someone calls the individual to say the sun is shining, more information is provided to the individual. The individual did not know if a storm or some fog had covered the shining sun. Using "events" helps ensure an OD system is not processing data that is already known.

The contribution of this research is an information-efficient and low-latency obstacle detection algorithm that combines these design principles:

- No complex classes (car, road, pedestrian, etc.)
- The usage of events

Since obstacle avoidance (OA) is the end goal of OD, the algorithm will then be analyzed to verify there is no reduction in obstacle avoidance performance. To improve the algorithm's robustness, modifications will be presented to eliminate the need of vehicle state measurements external from the algorithm.

1.1 Background

Obstacle detection is "the determination of whether a given space is clear from obstructions for safe travel by an autonomous vehicle", according to Singh and Keller [6]. This thesis will expand this definition slightly to include any vehicle, manned or unmanned. A variety of sensors, such as LiDAR, RADAR, ultrasonic, visible-light cameras, and infrared cameras, are used for OD in autonomous vehicles. As processing power increases and technology matures, LiDAR sensors have become popular for their wide field of view, direct range measurements, and resolution. The processing pipeline for a LiDAR OD system often contains data association algorithms for segmenting the scene into relevant areas [4,7]. Typical relevant areas include ground, non-ground, and obstacle areas. These algorithms can be as simple as a height threshold or nearest-neighbors clustering, or as sophisticated as deep neural networks.

One of the rising issues of OD is the ability to process large amounts of information in real time [8]. Sensor manufacturers continually create devices that provide more data to end users. However, this data must be efficiently and selectively filtered to maintain real time requirements without decreasing data quality. Data association algorithms can be computationally intensive and often do not provide information about the object's obstacle status.

Neuromorphic sensing and processing are potential solutions for efficiently handling large amounts of information. Event cameras, or dynamic vision sensors (DVS), are a relatively new technology that mimic a human retina. Lichsteiner et. al. developed one of the first DVS, which possesed the high dynamic range, low-latency, and event-based characteristics of the human visual system [9]. One of the most distinguishing features of DVS is the output rate of each pixel is controlled locally and is independent of surrounding pixels. Due to the low-latency, event cameras do not experience motion blur in high-speed driving scenarios [10]. Liu and Delbruck emphasize one of the advantages of using neuromorphic sensory systems is transmitting "only informative non-redundant events" [11]. Despite the desirable attributes of neuromorphic systems, there are substantial barriers to deploying

neuromorphic systems. These barriers include spatiotemporal and photometric differences that are not compatible with traditional frame-based sensors and algorithms [12]. The development of neuromophic, visible-light cameras is well-underway; however, neuromophic LiDAR still remain largely unexplored. Vyas created a silicon retina for processing LiDAR time-of-flight (ToF) measurements to detect object motion in autonomous ground vehicle applications [13]. Despite showing low-latency, a custom chip for processing LiDAR data is not a viable solution for sensor system manufacturers, who are looking for cheap and flexible processing solutions.

To obtain the "best of both worlds", a LiDAR system should be able to create non-redundant information with a commercially available processor and LiDAR sensor. Tsiourva and Papachristos create a LiDAR saliency map by fusing information from visible, intensity, reflectivity, and range images obtained from an Ouster OS1-64 sensor [14]. This approach does create a saliency map, but does not utilize events to reduce the available information to the most important elements. Singh et. al. create a confidence map from a LiDAR sensor for small road obstacles by finding breakpoints in the LiDAR scans, projecting the breakpoints to an RGB image of the scene, and smoothing the confidence values with a Gaussian kernel [15]. Again, the idea of the saliency map is there but the non-redundant, event-based elements are not. The focus of this thesis is to develop an OD system from a LiDAR point cloud using an event-based saliency map created with commercially available processors and sensors.

1.2 Objectives

There are three primary objectives:

1. Design and test an event-based OD system that does not use complex classes
2. Verify there is enough information in the output of the OD system for performing OA
3. Remove dependencies on the OD system by estimating vehicle states such that the output of the OD system is still accurate

1.3 Methods

1.3.1 Event and Importance Maps

The OD algorithm will utilize range events, which are binary signals that capture a change in range of subsequent LiDAR returns. An event map is an image showing the presence or absence of range events over the LiDAR's FOV. However, range events do not always indicate the presence of an obstacle. Changes in the scene, changes in the ego-vehicle's[1] orientation, and noise all contribute to the creation of range events. To limit the effects of these factors in obstacle detection, another processing step, including low-pass filtering and static object tracking, creates a non-binary image depicting where obstacles most likely exist. This image is called an importance map. Obstacles are identified by thresholding the importance map to create a binary image, or an obstacle mask.

1.3.2 Verification of Task Information Preservation

As described in the first paragraphs of Chapter 1, the amount of information present can be measured quantitatively. However, this measure does not account for low-probability outcomes (outcomes with high information content) that are irrelevant for the execution of a particular task. Extrapolating from the example in the beginning of this chapter, if the individual inside the building is going to be working inside all day, a weather report does not affect the individual's work. On the other hand, if the individual needs to work outside for part of the day, a weather report provides valuable information. This presents the notion that the required information is less than the available information. To verify information is not lost when using the importance map, a convolutional neural network (CNN) designed for OA tasks in autonomous driving applications will be trained using standard LiDAR data and importance map data. If the CNN continues to perform well when trained with the importance map versus more typical data modalities, such as a range or intensity image, the importance map is not discarding valuable information.

[1] The ego vehicle is the vehicle-under-test, or the vehicle using the obstacle detection system.

1.3.3 Vehicle State Estimation

The creation of the importance map requires vehicle state measurements, vehicle longitudinal velocity and angular rate. A system external to the OD system can provide these measurements; however, installation and hardware costs will increase. To make the user's interaction with the OD system more simple, the vehicle state measurements will be estimated using the available LiDAR data and event maps.

Gallego et. al. have successfully estimated angular rate and velocity from event images (images obtained from event-based cameras) by warping the event image such that the contrast is maximized [16, 17]. In event images, motion blur is easily seen because event images have millisecond resolution. Since event maps are only updated every tenth of a second, there is no motion blur. Vehicle state estimation for the importance map will utilize the movement of static objects in the LiDAR frame instead of motion blur to determine vehicle longitudinal velocity and vehicle angular rate. Static object movement will enable a mechanism for finding corresponding LiDAR returns in subsequent scans. The vehicle state measurements can then be obtained using point registration methods and estimation algorithms that utilize the vehicle's dynamic model.

1.4 Contributions

The principle contributions of this work are:

- A novel OD system using range events created from LiDAR returns (Chapter 2)
- A unique temporal filter for processing binary signals (Chapter 2)
- Showing range events are highly discriminatory of obstacles (Chapter 2)
- Showing range events can make an OD system more computationally efficient (Chapter 2)
- Showing event and importance maps contain sufficient information for obstacle avoidance (Chapter 3)

- A unique approach for weighting point correspondences in point registration problems when using a classifier (Chapter 4)
- A unique method for simultaneously estimating vehicle velocity and angular rate using only LiDAR data with $O(n)$ calculations for finding matching points (Chapter 4)

Another atypical element of this work, that is not novel, is the simplification of the OD problem: limiting obstacles to objects that are on a collision course with the vehicle (see first paragraphs of Chapter 2). This simplification enables the effectiveness of the OD system presented in this work. Defining an obstacle this way is advantageous for typical computing platforms as well as understandable by a human. Using complex classes, such as car and pedestrian, are not easily calculated on computers but they are very obvious and simple for humans. Collision-course obstacles can be directly measured by range sensors (i.e. LiDAR) and easily operated on by computers.

CHAPTER 2

OBSTACLE DETECTION WITH THE IMPORTANCE MAP

The importance map is a biologically inspired method for obstacle detection. The importance map is not a neuromorphic system because the importance map does not emulate and mimic a human's visual processing pathway as close as possible. Instead, the importance map utilizes conventional computing architectures (i.e. synchronous and digital computations in a pipeline) along with ideas from biology, such as using events to convey information. It is important to realize that modern obstacle detection systems should not try to emulate every aspect of a biological OD system, but rather take the best from both worlds [18].

The importance map is an ideal candidate for emergent obstacle detection because the importance map is calculated using events from directly sensed features (i.e. range) instead of complex, human-defined classifications. Instead of relying on obstacles being any hazardous object, the term obstacle will be limited to any object that will collide with the ego vehicle if the ego vehicle and obstacle do not change their velocities or path. Furthermore, the importance map is LiDAR-centric, meaning all calculations occur in the LiDAR's frame of reference. This makes the importance map ideal for distributed obstacle detection systems where obstacle detection algorithms are executed on processors within the sensor. The importance map is also an expert system, which means no training is required. This allows the importance map to be used in a larger variety of environments without worry of encountering scenes that are out of the training data's distribution.

2.1 Theory

LiDAR point cloud processing in obstacle detection applications requires several stages where the number of points that could correspond to obstacles in the scene decreases upon the completion of each stage. For example, if an obstacle detection pipeline performs ground

segmentation in the first stage, all points are candidates for obstacles before this stage. After the ground segmentation, all points that are classified as ground are no longer considered. The motivation for event-based LiDAR point cloud processing is to eliminate as many points as quickly as possible.

Let N_i designate the number of eligible points at the beginning of each stage i, and let C_i be the average computational cost per point in stage i. Therefore, the computational cost for the obstacle detection system can be described as

$$C = \sum_{i=0}^{K-1} N_i C_i = N_0 \bar{C}, \tag{2.1}$$

where C is the total cost, K is the number of stages, N_0 is the total number of points, and $\bar{C}$ is the average computational cost per point. An important property of (2.1) is $N_{i-1} \geq N_i$ since no additional points are generated. This implies the most efficient obstacle detection systems have $N_{i-1} \gg N_i$ and the computations ordered such that $C_{i-1} \leq C_i$. When an obstacle detection system is organized in this configuration, total computations are always decreased by processing and eliminating more points in the earlier stages when computations per point are cheap. Another stage can be added in an attempt to reduce the cost in an existing pipeline:

$$C = N_0 \bar{C} \overset{?}{>} C' = N_0 \bar{C}' = N_0 C_{0+} + N_{0+} \bar{C}. \tag{2.2}$$

The new stage is added by inserting another stage of computation C_{0+} that reduces the number of remaining points to N_{0+} before continuing with the rest of the computations. The cost associated with the new pipeline is C'. If C and C' are equivalent, Equation (2.2) can be expressed as

$$C = N_0 \bar{C} = N_0 (a\bar{C}) + (bN_0)\bar{C} \implies 1 = a + b, \tag{2.3}$$

where

$$a = \frac{C_{0+}}{\bar{C}} \text{ and } b = \frac{N_{0+}}{N_0}. \tag{2.4}$$

The ratio a compares the computations in the new stage with the computations in the old stage, and the ratio b compares the remaining number of points after the new stage with the number of points remaining before the old stage. From (2.3), the total computational cost will remain the same as long as $(a + b) = 1$. The total computational cost will only be reduced if $(a + b) < 1$.

Biological and neuromorphic systems have negligible overhead when creating events because the systems are usually analog devices, where operations happen at the speed of light with a short propagation delay. Implementing events in a digital system requires an overhead cost associated with a processor's instruction pipeline and memory access, which cannot be ignored. Using events in an obstacle detection system is the same as adding another stage, as shown in (2.2). Since events are relatively easy to compute, such that $a < 1$, and usually highly discriminatory, such that $b \ll 1$, using events can make a digitally-implemented OD system more efficient. Section 2.2 shows the ratio b is approximately 0.1 in practice.

2.1.1 Importance Map

An importance map is an 8-bit, single-channel image where the value of each pixel represents the "importance", or likelihood the object located at the pixel coordinate is an obstacle. Using the Ugly Duckling Theorem [19], a pixel is only deemed important if the pixel is classified as important according to "importance features", which are selected beforehand or *a priori*. The importance features selected in this work are event, constant-angle, "in vehicle path", and static object, where the output of each feature calculation is a boolean value. These features were selected for their pixel-wise calculability and ability to discriminate obstacles from non-obstacles.

After feature calculations, the results are combined using a boolean expression, giving a single boolean output indicating if a pixel is important or not. Since this boolean output

is noisy, the output is filtered using the multi-response filter (MRF) [20]. The quantized real-valued output of the MRF is the importance map value at a single pixel.

The descriptions in this section are taken from previous work [20].

Event Features

An event map is a binary image representing a change in range between subsequent LiDAR returns at the same azimuth and elevation (θ, ϕ) coordinate in a LiDAR scan. A LiDAR scan can be represented as an image where each pixel value is a range measurement. The value θ is directly proportional to the column location in the scan, and ϕ is directly proportional to the row location. Let

$$\Delta R_{\theta,\phi}(t) = R_{\theta,\phi}(t-1) - R_{\theta,\phi}(t), \tag{2.5}$$

such that $R_{\theta,\phi}(t)$ is a range measurement from a LiDAR return. Using Iverson's notation [21], the event map can be expressed as

$$E_{\theta,\phi}(t) = [R_{min} \leq \Delta R_{\theta,\phi}(t) \leq R_{max}], \tag{2.6}$$

where the value of the event map at (θ, ϕ) is 1 when the expression within the brackets is true. R_{min} is always greater than 0 since an object will never be an obstacle if the distance between the object and the ego vehicle is increasing. The value of R_{min} dynamically changes with respect to $R_{\theta,\phi}(t)$ to accommodate an increase in range measurement variance as the signal-to-noise ratio (SNR) decreases [22].

Constant-Angle Features

Constant-angle features follow the constant-angle principle. The constant-angle principle states a collision will occur between two moving objects if the distance between the two objects along a constant angle, with respect to a reference direction, is decreasing. Chance uses this idea to pre-program a neural network to simulate a dragonfly's ability to catch prey [23]. Figure 2.1 shows the constant-angle principle in action.

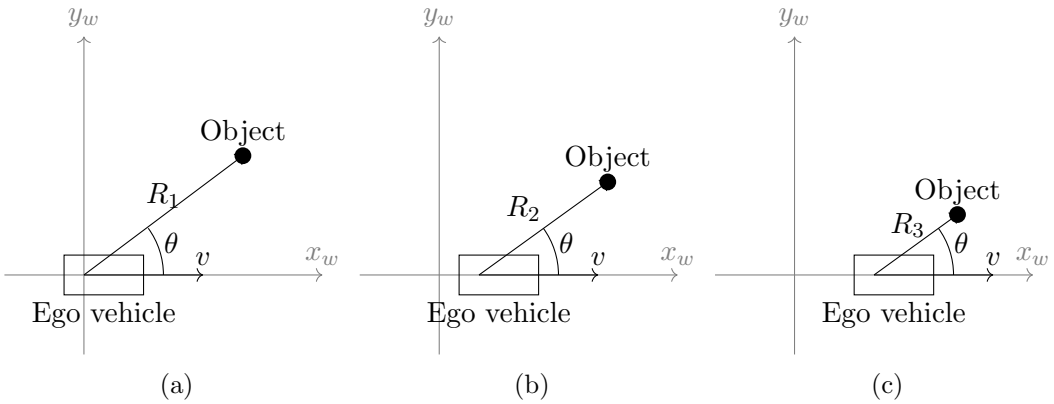


Fig. 2.1: Diagram depicting the constant-angle principle, where $R_1 > R_2 > R_3$ and θ is constant. Subfigures (a), (b), and (c) occur in chronological order.

The constant-angle principle is a powerful idea that allows moving agents to interact with objects without precisely knowing the object's location or motion characteristics. This idea describes how humans and animals are able to avoid or intercept moving targets without knowing much about the target. However, the constant-angle principle is not an all-inclusive description of obstacle collision, but it provides a good approximation for the motion of most obstacles. Motion along a constant angle is easily detected by testing for subsequent events occurring in the same coordinate location in the event map.

The constant-angle principle does enforce LiDAR mounting constraints such that an obstacle collision with the LiDAR sensor is inclusive of obstacle collisions with the ego vehicle. To relax this requirement, instead of only using the previous event at the same coordinate location as the current event, the Moore neighborhood (set of grids cells with adjacent sides or corners of a given cell) of the previous event is also searched.

In Vehicle Path Features

The "in vehicle path" feature determines if an event was caused by an object within the Area of High Importance (AHI) [22]. The AHI indicates a likely area for the vehicle's future path. This area is represented by a parallelogram-like polygon extending from the front of the vehicle that slants according to the vehicle's travel curvature, as shown in Figure 2.2.

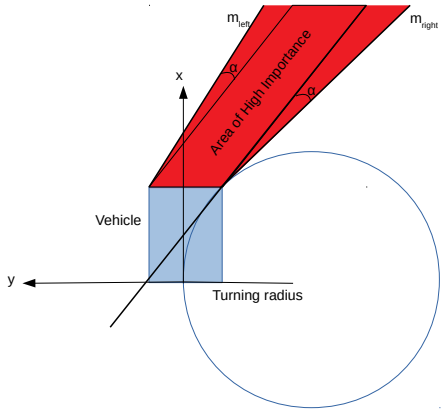


Fig. 2.2: Diagram showing the AHI with respect to the ego vehicle.

The slopes m_{left} and m_{right} in Figure 2.2 are

$$m_{left} = \tan(l\frac{\omega_v}{v_x} + \alpha) \tag{2.7}$$

and

$$m_{right} = \tan(l\frac{\omega_v}{v_x} - \alpha), \tag{2.8}$$

where l is the wheelbase of the vehicle, ω_v is the yaw rate of the vehicle, v_x is the longitudinal velocity of the vehicle, and α is an expansion angle. The left and right boundaries of the AHI are calculated using point-slope form with the AHI's anchor points in the LiDAR frame (such as the front corners of the vehicle) and the boundaries' respective slopes.

Static Object Features

Static object features label events as either corresponding to a static object or not. Prior work reports static object tracking to prevent information from static objects reaching the importance map since a static object not in the road can never be an obstacle [22]. When an object is static in the world frame, the object moves in a predictable manner in the vehicle and LiDAR frames when the ego vehicle is moving in the world frame. Static object

tracking is the determination of a static object's position in the vehicle or LiDAR frame at a previous time step in polar coordinates. Franceschini et. al. constructed an insect-inspired robot that calculated its distance from obstacles using the two-dimensional, instantaneous motion parallax equation:

$$\dot{\theta} = \frac{v}{R}\sin(\theta), \tag{2.9}$$

where v is ego vehicle velocity, R is the distance to the static object, and θ is the angle from the ego vehicle to the static object [24]. Figure 2.3 illustrates (2.9). Cornwall et. al. also use (2.9) to do static object tracking [22].

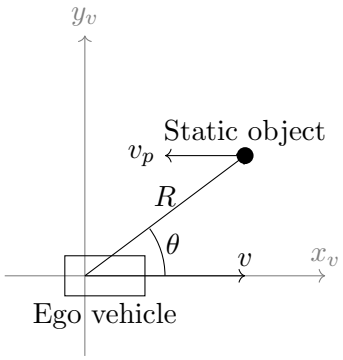


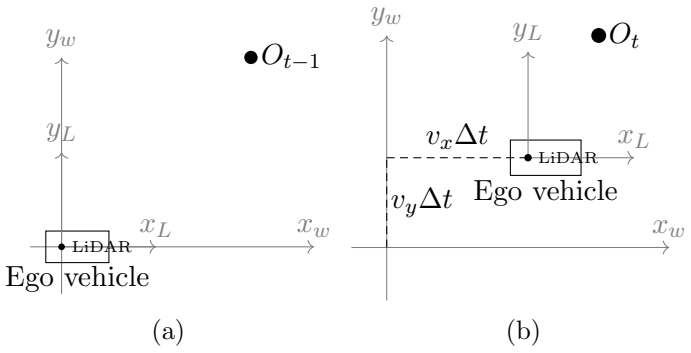
Fig. 2.3: Diagram depicting the two-dimensional physical situation where (2.9) applies. The perceived velocity of the static object from the ego vehicle is shown as v_p.

The principle limitation when using (2.9) for static object tracking is the assumption of a constant static object distance R at times t and $t-1$. At lower speeds, this assumption holds, and variations of (2.9) can be used effectively. However, as the vehicle's speed increases, $|R_t - R_{t-1}|$ increases. Furthermore, $\dot{\theta}$ in (2.9) is not constant during a time interval because R and θ are also changing as the vehicle drives closer to or past the static object. In order to use (2.9) to find the previous angular position of the static object at a previous time $t-1$, the following integral would have to be calculated:

$$\Delta\theta = \int_{t-1}^{t} \frac{v}{R(t)}\sin(\theta(t))dt, \tag{2.10}$$

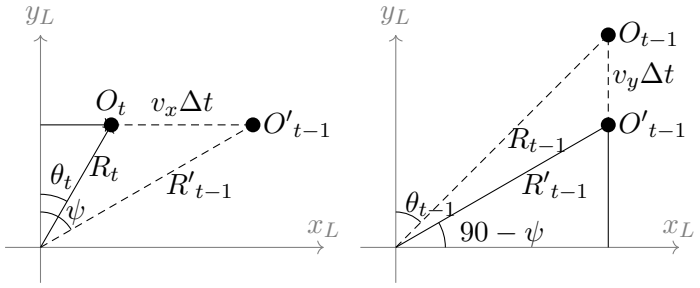
where $R(t)$ and $\theta(t)$ are coupled. This coupling makes the integral difficult to calculate. To avoid calculating (2.10), a method using similar triangles is presented.

Figure 2.4 shows the static object motion in the x-y plane of the world frame. Assume the vehicle's longitudinal and angular velocity remains constant over the interval $[t, t-1]$. From this assumption, Figure 2.5 describes the relationship between a static object at time t and the same static object at time $t-1$ in the x-y plane of the LiDAR frame.



(a) (b)

Fig. 2.4: Diagrams showing the position of the LiDAR and the static object in the x-y plane of the world frame when the vehicle is moving. (a) Vehicle at time $t-1$. (b) Vehicle at time t.



(a) (b)

Fig. 2.5: Diagrams depicting the geometric relationship between a static object at t and the same static object at $t-1$ in the LiDAR frame. The ranges R shown are the ranges in the x-y plane. (a) Static object movement in the LiDAR frame in the x direction, as shown in 2.4(b). (b) Static object movement in the LiDAR frame in the y direction.

The variables ψ, R'_{t-1}, and O'_{t-1} in Figure 2.5 represent intermediate values used to find θ_{t-1}. Using Figures 2.5(a) and 2.5(b),

$$\psi = \tan^{-1}\left(\frac{v_x \Delta t}{R_t \cos(\theta_t)} + \tan(\theta_t)\right). \tag{2.11}$$

and

$$\theta_{t-1} = -\tan^{-1}\left(\frac{v_y \Delta t}{R'_{t-1}\cos(90-\psi)} + \tan(90-\psi)\right) + 90. \tag{2.12}$$

It is important to note the range variables R_t and R'_{t-1} are two-dimensional distances in the x-y plane in (2.11) and (2.12).

Using the same relationships shown in (2.11) and (2.12), the equations for finding the previous β (pitch) location of a static object in the x-z plane are

$$\zeta = \tan^{-1}\left(\frac{v_x \Delta t}{R_t \cos(\beta_t)} + \tan(\beta_t)\right) \tag{2.13}$$

and

$$\beta_{t-1} = -\tan^{-1}\left(\frac{v_z \Delta t}{R'_{t-1}\cos(90-\zeta)} + \tan(90-\zeta)\right) + 90, \tag{2.14}$$

where ζ is the intermediate angle (analogous to ψ) and R is now the distance of the object in the x-z plane. Appendix A.1 provides explicit figures for the static object geometric relationships in the x-z plane.

The vehicle velocity terms v_x, v_y, and v_z are scalars representing the vehicle's velocity expressed in the vehicle frame. Since an object's pitch location β_t is not usually directly available from the LiDAR, β_t can be obtained from the azimuth θ_t and elevation ϕ_t coordinates of a LiDAR return with

$$\beta_t = \tan^{-1}\left(\frac{\cos(\theta_t)}{\tan(\phi_t)}\right), \tag{2.15}$$

and ϕ_{t-1} can be obtained from $(\theta_{t-1}, \beta_{t-1})$ coordinates with

$$\phi_{t-1} = \tan^{-1}\left(\frac{\cos(\theta_{t-1})}{\tan(\beta_{t-1})}\right). \tag{2.16}$$

Equation (2.16) allows the static object tracking algorithms to find the previous elevation location ϕ_{t-1} of an object from θ_{t-1} and β_{t-1}. Algorithms 1 and 2 in previous work [20] provide explicit instructions on how to implement the static object tracking equations to find the proposed previous angular location of the object. If the proposed previous angular location of the object (pixel) is classified as an event, then the static object feature is present.

Temporal Filtering

The event map accumulates noise from abrupt changes in the vehicle's orientation, changes in the scene, and noise in the LiDAR returns. There are two basic types of filtering: spatial and temporal. Spatial filtering uses pixels at the same time step around the pixel of interest. Temporal filtering uses pixels at the same image coordinate, but at different time steps. In an event-based system, spatial filtering is inefficient because all pixels in an image must be completely processed before noise removal can occur. Temporal filtering permits noise removal without waiting for adjacent pixels to be processed. In this application, temporal filtering can also promote and penalize values that likely correspond to obstacles and non-obstacles, respectively. Therefore, filtering must remove noise in all cases but respond differently when faced with obstacles and non-obstacles.

The multi-response filter (MRF) temporally processes binary signals such that rising edges and falling edges have unique responses. For example, when a value in the importance map is likely an obstacle (binary 1), the value's importance should increase quickly and decrease slowly. When a value in the importance map is a likely non-obstacle (binary 0), the value's importance should decrease quickly and increase slowly. Let x_t be the input binary signal indicating if the value is likely an obstacle, and let y_t be the new importance value. The MRF is a mapping $\{x_t : x_t \in \{0,1\}\} \rightarrow \{y_t : y_t \in [v_{ff}, v_{fr}], y_t \in \mathbb{R}\}$ and can be described by

$$y_t = [x_t a_r + (1 - x_t)a_f]y_{t-1} + x_t c_r + (1 - x_t)c_f, \tag{2.17}$$

where Table 2.1 provides descriptions for the variables in (2.17). Equation (2.17) can also be represented as

$$y_t = \begin{cases} a_r y_{t-1} + c_r, & x_t = 1 \\ a_f y_{t-1} + c_f, & x_t = 0 \end{cases}. \tag{2.18}$$

Table 2.1: Variables used in the MRF

Variable	Domain	Description
a_r	$\{[0,1] : a_r \in \mathbb{R}\}$	Parameter inversely related to the rising rate of the output
a_f	$\{[0,1] : a_f \in \mathbb{R}\}$	Parameter inversely related to the falling rate of the output
c_r	$\{[0, v_{fr}] : c_r \in \mathbb{R}\}$	Value that controls the rising-edge final value
c_f	$\{[0, v_{ff}] : c_f \in \mathbb{R}\}$	Value that controls the falling-edge final value
v_{ff}	$\{(-\infty, \infty) : v_{ff} \in \mathbb{R}\}$	Parameter specifying the rising-edge final value
v_{fr}	$\{(-\infty, \infty) : v_{fr} \in \mathbb{R}\}$	Parameter specifying the falling-edge final value

To determine the values of c_r and c_f that satisfy v_{fr} and v_{ff}, respectively, the final value theorem is applied to the unilateral Z-Transform of (2.18) while modeling c_r and c_f as unit step functions. This analysis shows

$$v_{fr} = \frac{c_r}{1 - a_r} \text{ and } v_{ff} = \frac{c_f}{1 - a_f}, \tag{2.19}$$

which bounds y_t to the range $[v_{ff}, v_{fr}]$ as long as y_0 is also in the range $[v_{ff}, v_{fr}]$. Rearranging (2.19) shows how c_r and c_f are calculated using MRF parameters.

The value a_r is initialized in the range $[0, 1]$ where values closer to 1 cause slower rising rates and values closer to 0 cause faster rising rates. The new a_r is obtained by scaling the previous a_r by the normalized range, where the values $[R_{min}, R_{max}]$ are linearly mapped to the range $[0, 1]$. Before updating y_t, (2.19) is used to update c_r.

The binary input x_t is created by fusing the importance features for each pixel using the boolean expression

$$x_t = (\text{event})(\text{constant-angle})((\text{in vehicle path}) + NOT(\text{static object})), \tag{2.20}$$

where multiplication and addition correspond to *AND* and *OR* operations, respectively. When a LiDAR return is within the AHI, a_r is dynamically adjusted. This helps reduce false-positives caused by the road (Section 2.2.4 addresses this in more detail).

An alternative to the MRF is to model the problem using probabilities, such as a discrete Bayes filter or Hidden Markov Model [25, 26]. Probabilities are ideal in situations where knowing the most likely state is sufficient. In other words, probabilistic filters give the best estimate of the true state of the environment. This is not good enough in some applications.

An example is obstacle detection. When there is an extremely high cost for failure, such as property damage and loss of life, only reporting objects that are most likely obstacles is not sufficient. One such scenario is when an obstacle is close to the vehicle. In this situation, stopping for a false obstacle is much better than colliding with a true obstacle. The MRF is useful in these situations because it does not try to give a best estimate of the true state, but rather enforces a behavior specified by the user. This behavior is enforced by dynamically adjusting the rising and falling rates, a_r and a_f, according to a user-defined model. When executing the correct or safe behavior is a higher priority than knowing the best estimate of the true state of the environment, the MRF is an ideal choice. An OD system is dealing with a single instance of an object for a few fractions of a second. An obstacle, by definition, is something not planned for, which can make probabilistic modeling of obstacles difficult. The MRF allows the engineer to ensure specific responses under specific conditions that are not easily modelled using probability theory.

2.1.2 Obstacle Detection Algorithm

Point obstacles in the scene, or pixels in the importance map that likely correspond to obstacles, can be visualized by creating an obstacle mask. An obstacle mask is a binary

image created from the importance map using a predefined, global threshold where all values above the global threshold are considered obstacles. This binary mask is a natural extension from the importance map: once a point reaches a certain level of importance, the point should be considered an obstacle.

Algorithm 1 shows the entire process from LiDAR returns to the obstacle mask.

Algorithm 1 Obstacle Detection

Input: LiDAR return, Vehicle Dynamics ▷ Quantities expressed in LiDAR frame

```
for Each LiDAR return do
    if return is an event then
        if event is constant-angle then
            if event is in vehicle path then
                imp ← MRF(x_t ← 1, a_r ← normalized range)
            else if event is not static object then
                imp ← MRF(x_t ← 1)
            else
                imp ← MRF(x_t ← 0)
            end if
        else
            imp ← MRF(x_t ← 0)
        end if
        Place imp in importance map at (row, col)
        if imp ≥ threshold then
            Place 1 in obstacle mask at (row, col)
        else
            Place 0 in obstacle mask at (row, col)
        end if
    end if
end for
return importance map and obstacle mask
```

Using observations made in the first paragraphs of Section 2.1, the least computationally intensive and most discriminatory features are calculated in order: event, constant-angle, "in vehicle path", and static object. Since the static object feature is the most

expensive, the feature is not calculated until the pixel has been labelled with the other three features.

2.2 Experiment

The importance map used in previous work [22] and the improved importance map as presented here were compared using LiDAR scans from the KITTI data set [27]. The significant differences between the two importance maps are: the old importance map does not use the constant-angle feature nor the MRF, and the new importance map uses an improved method for calculating the static object feature. The importance map is designed to work with LiDAR that report ordered point clouds, or point clouds where each point is a pixel in a 2D grid with the value of each pixel representing a range measurement. The KITTI data set only contains unordered point clouds, so the row/column location is explicitly calculated from the azimuth and elevation (θ, ϕ) values associated with each point.

The results compare the output from the two importance maps at three different scenes. Using the definition of an obstacle from the first paragraphs of this chapter, the KITTI data set does not have labels that identify obstacles. To evaluate the performance of the importance map algorithms, obstacles are established in the scene and the approximate number of true positives and false positives in the corresponding obstacle masks are compared. The global threshold for the obstacle mask was set to 150.

True positives are white points in the obstacle mask inside the obstacle bounding-box (red square) and on the vehicle that is inside the bounding-box. The vehicle inside the bounding-box is the target that must be identified by the OD system. False positives are white points in the obstacle mask that are not on the target. The goal is to have a system with a true-positive rate of 1.0 (every obstacle point is classified as an obstacle) and a false-positive rate of 0.0 (every non-obstacle point is classified as a non-obstacle).

The descriptions, results, and comments involving the images of the importance maps and obstacle masks were taken from previous work [20].

2.2.1 Drive 0084

The scene shown in Figure 2.6 is taken from the file 2011_09_26_drive_0084 in the KITTI data set at second 35. The ego vehicle is slowing down for a vehicle stopped at an intersection. The vehicle stopped at the intersection is the obstacle. In this example, the number of false positives drastically decreases and the number of true positives decreases. The reduction in true positives is not too much of a concern because binary image operators, such as morphological filters, can fill in the spaces between the true positives.

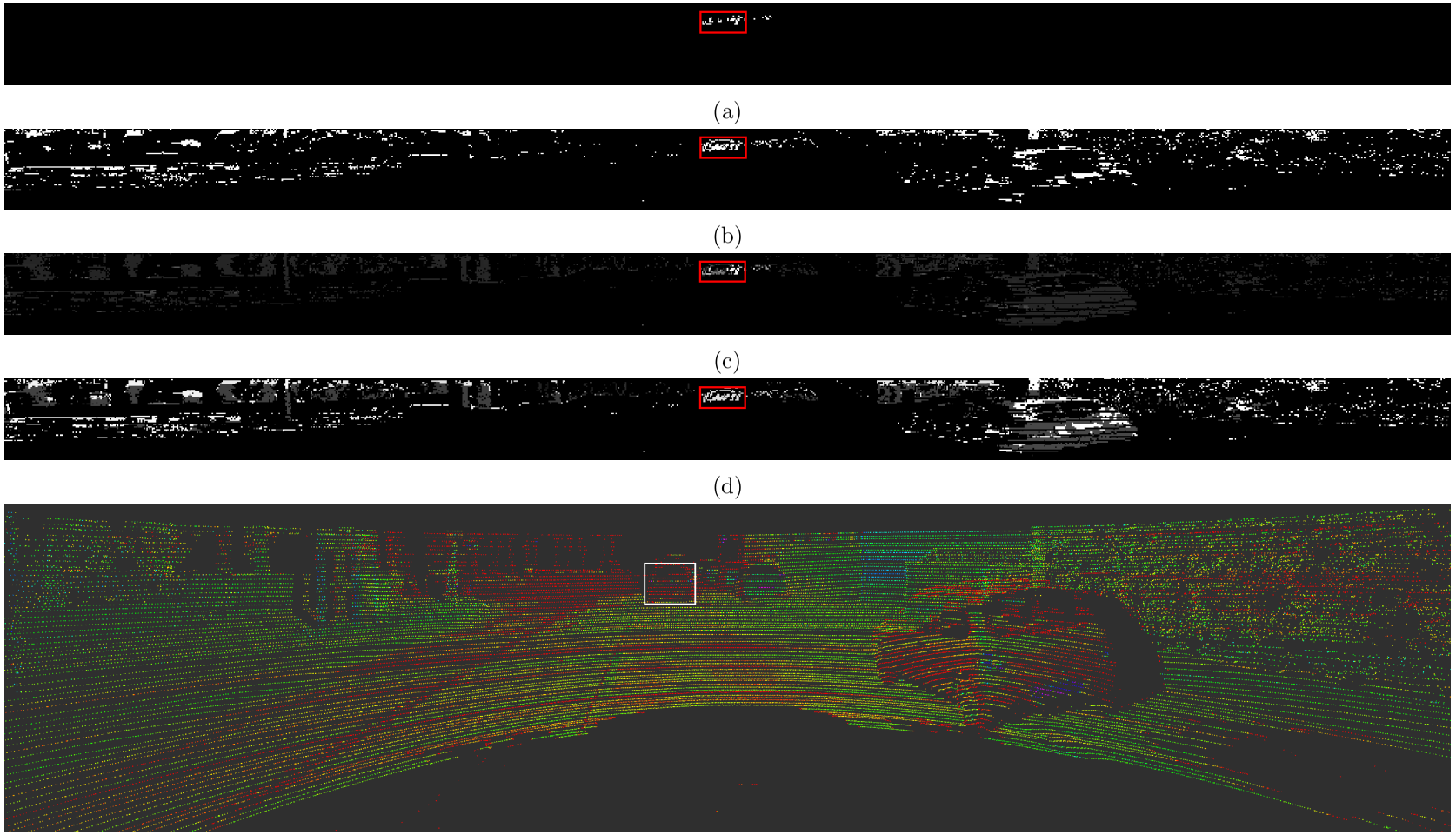
From the obstacle mask in Figure 2.6(a), the algorithm is able to identify the pixels in the scene that correspond to obstacles. This shows the features in the importance map are features associated with obstacles in this scenario. Table 2.2 shows ratios with respect to the total number points. Non-ground points in Table 2.2 are points at a height greater than the vehicle's axle. This is an example of ground segmentation, which is the first step in many OD systems [28]. Events are considered points that possess the event feature as described in Section 2.1.1. Important events are points that satisfy (2.20) with true. Table 2.2 shows the event ratio as approximately 4 times smaller than the non-ground ratio. Since the event ratio is smaller than the non-ground ratio, events are a better discriminator for obstacles in this data set.

Table 2.2: Ratio to Total Points in 0084 Drive

Ratio	New	Old
Non-ground	0.331	0.331
Events	0.088	0.088
Important Events	0.003	0.03

2.2.2 Drive 0042

The scene shown in Figure 2.7 is taken from file 2011_10_03_drive_0042 in the KITTI data set at second 19.5. The ego vehicle is on a highway about to pass vehicles that are entering the highway. The vehicle entering the highway is the obstacle. In this example, the number of false positives drastically decreases and there also appears to be a significant



(a)

(b)

(c)

(d)

(e)

Fig. 2.6: Figures showing the scene, importance maps, and obstacle masks for drive 0084. The white and red boxes show the obstacle's bounding box. (a) The obstacle mask created from the improved importance map. (b) The obstacle mask created from the old importance map. (c) The improved importance map. (d) The old importance map. (e) The scene as a LiDAR point cloud, colored according to intensity.

reduction in true positives.

The obstacle in Figure 2.7 is an object not directly in front of vehicle, which shows the constant-angle principle. Only the front of the vehicle shows as an obstacle because static object tracking is a pixel-wise operation and does not use an object's size during calculation. The other cars in the scene are ignored because they are not moving towards the ego vehicle. Even though the true-positive rate is reduced in the improved importance map, this example shows the importance map is not simply ignoring all points that are not in the vehicle's direction of travel. The importance map uses the criteria in (2.20) to adaptively ignore or attune to areas in the scene instead of using geometric boundaries.

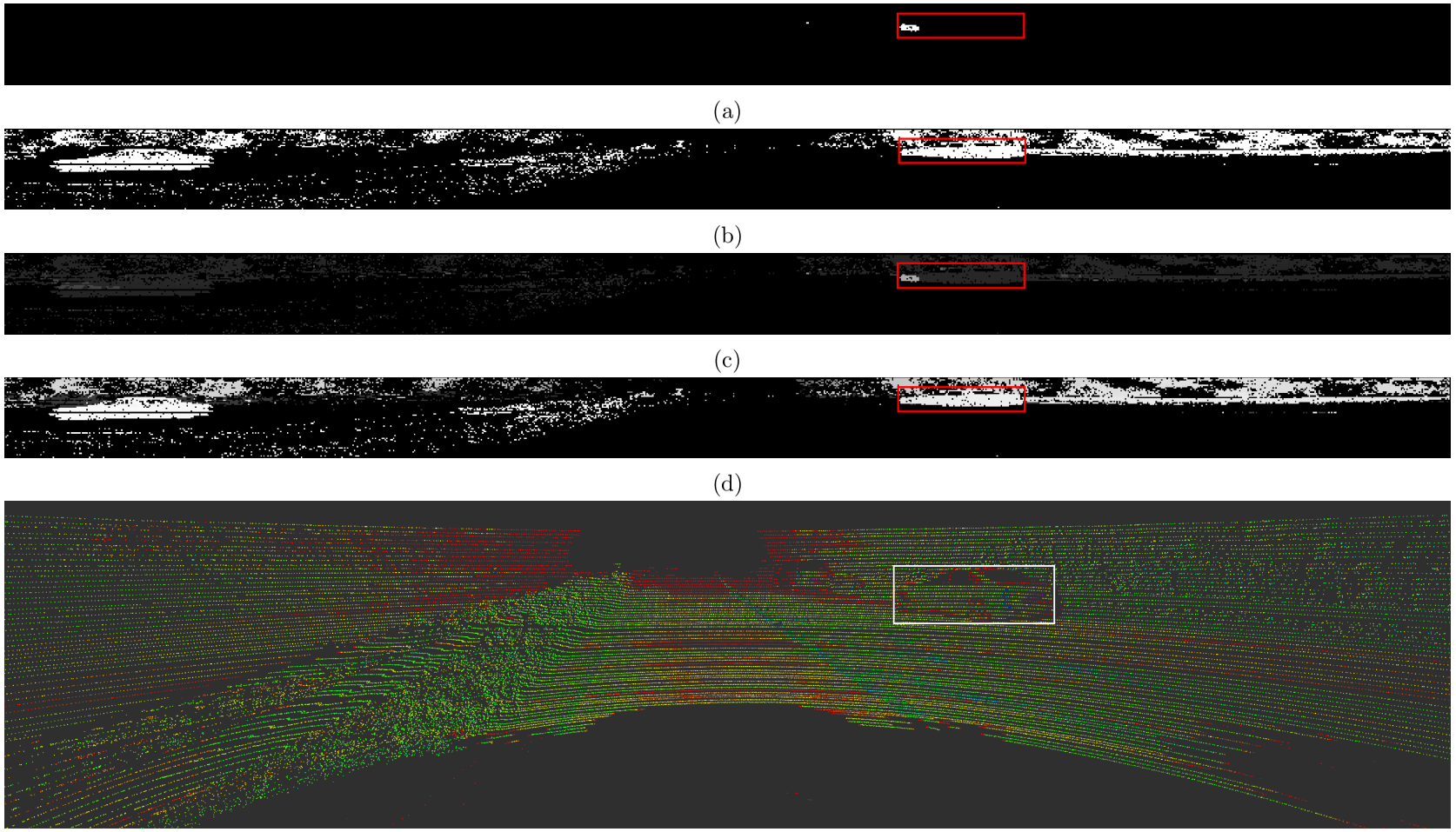
From the obstacle mask in Figure 2.7(a), the algorithm is able to identify the pixels in the scene that correspond to obstacles. This shows the features in the importance map are features associated with obstacles in this scenario. Table 2.3 shows ratios with respect to the total number points. The event ratio is approximately 3 times smaller than the non-ground threshold. Therefore, events remain a better discriminator for obstacles in this data set. Events experience a reduction in obstacle discrimination, as compared to non-ground thresholding, when the number of above-ground objects is small. In highway environments, there are less above-ground objects, explaining the smaller difference between the non-ground and event ratios.

Table 2.3: Ratio to Total Points in 0042 Drive

Ratio	New	Old
Non-ground	0.251	0.251
Events	0.079	0.079
Important Events	0.003	0.045

2.2.3 Drive 0022

The scene shown in Figure 2.8 is taken from the file 2011_09_26_drive_0022 in the KITTI data set at second 56. The ego vehicle is driving through a residential area when an oncoming vehicle turns into the lane. The vehicle entering the lane is the obstacle. In this



(a)

(b)

(c)

(d)

(e)

Fig. 2.7: Figures showing the scene, importance maps, and obstacle masks for drive 0042. The white and red boxes show the obstacle's bounding box. (a) The obstacle mask created from the improved importance map. (b) The obstacle mask created from the old importance map. (c) The improved importance map. (d) The old importance map. (d) The scene as a LiDAR point cloud, colored according to intensity.

example, the number of false positives drastically decreases without a significant reduction in true positives. This scene from drive 0022 was included because it is one of the few moments in the KITTI data set where a head-on collision is imminent.

From the obstacle mask in Figure 2.8(a), the algorithm is able to identify the pixels in the scene that correspond to obstacles. This shows the features in the importance map are features associated with obstacles in this scenario. Table 2.4 shows ratios with respect to the total number points. The event ratio as nearly 4 times smaller than the non-ground threshold. Therefore, events remain a better discriminator for obstacles in this data set.

Table 2.4: Ratio to Total Points in 0022 Drive

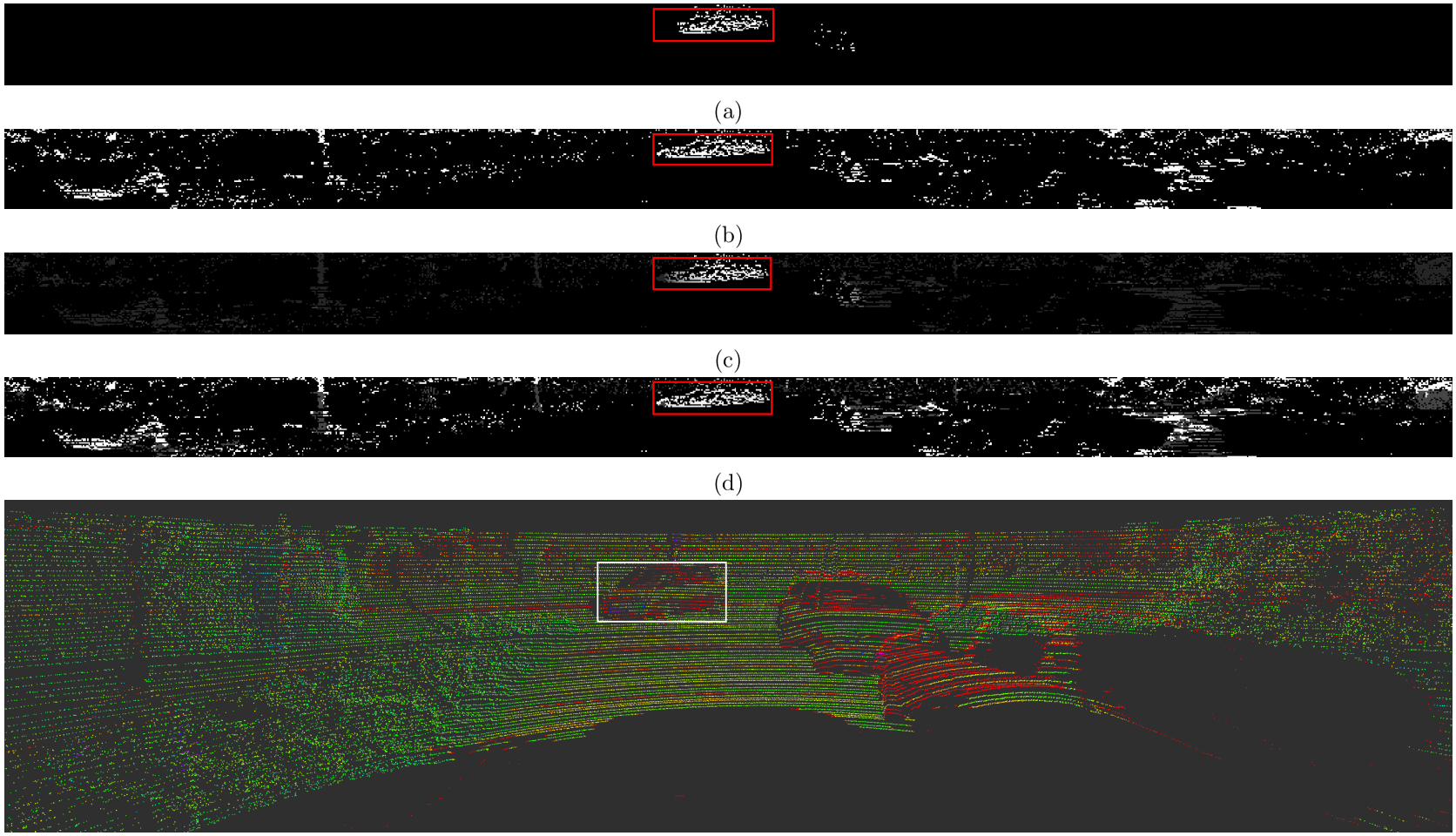
Ratio	New	Old
Non-ground	0.401	0.401
Events	0.106	0.106
Important Events	0.005	0.037

2.2.4 Comments

Due to the constraints mentioned at the end of Section 2.1.1, the LiDAR should be mounted relatively close to the ground. Mounting the LiDAR on top of small vehicles (such as a Volkswagen Passat in the KITTI data) works fine. However, if larger trucks are used, the LiDAR should be mounted somewhere close to the front grill.

When vehicles experience large pitch rates, such as travelling over a speed bump, the road is briefly reported as an obstacle. This occurs because pitch rates induce range events in the AHI. Further work will be needed to solve this problem efficiently.

Even though the importance map has been developed for autonomous ground vehicles, the importance map could be readily applied to a drone. Since a drone is a relatively small vehicle, the constraints for using the constant-angle principle are essentially non-existent. The importance map is also designed to find dynamic obstacles in all directions, which is necessary when road structures do not exist.



(a)

(b)

(c)

(d)

(e)

Fig. 2.8: Figures showing the scene, importance maps, and obstacle masks for drive 0022. The white and red boxes show the obstacle's bounding box. (a) The obstacle mask created from the improved importance map. (b) The obstacle mask created from the old importance map. (c) The improved importance map. (d) The old importance map. (e) The scene as a LiDAR point cloud, colored according to intensity.

2.3 Conclusion

The improved importance map creates an obstacle mask that correctly identifies obstacles in a scene without excessive false positive detections. Correct obstacle identification in Section 2.2 affirms the features presented in Section 2.1.1 are characteristic of obstacles. Additionally, events are computationally efficient and highly discriminatory, which can further reduce the computational costs of obstacle detection. Event-based obstacle detection using traditional processors and LiDAR sensors is a viable approach for efficient obstacle detection. The results in this chapter satisfy objective 1 as stated in Chapter 1.

Another implication of these results is the usability of the minimalist definition of an obstacle as presented in the first paragraphs of Chapter 2. This obstacle definition reduced the scope of the obstacle detection problem and facilitated the application of event, constant-angle, and static object features.

CHAPTER 3

VERIFICATION OF THE IMPORTANCE MAP

The importance map removes LiDAR returns that are not typically removed in traditional obstacle detection methods due to the minimalist obstacle definition presented in Chapter 2. Intuition supports that only paying attention to objects on a collision course with the ego-vehicle is sufficient for obstacle detection and avoidance. Instead of merely accepting this intuition, convolutional neural networks (CNNs) will be used to ensure valuable information for obstacle avoidance is not lost with the importance map. In this chapter, the primary focus will be on obstacle avoidance since the purpose of obstacle detection is obstacle avoidance.

3.1 Theory

First, a mathematical formulation of finding the minimum information for a task is presented using information theory. The CNN architecture used in the experiments is then described.

3.1.1 Finding the Minimum Information

Since the information content in data is directly connected to the uncertainty associated with the data, it is natural to model the problem of finding the minimum information using probabilities. The information, or entropy, associated with a discrete random variable $\mathcal{X}$ is designated as

$$H(\mathcal{X}) = -\sum_{x \in \Omega} p(x) \log p(x), \tag{3.1}$$

such that the logarithm is base-2, x is a realization of $\mathcal{X}$, and $p(x)$ is the probability of a realization occurring [5].

Let the result of a high-level task, such as OD, be denoted as

$$\mathcal{Z} = G(\omega) \; s.t. \; \omega \in \Omega, \tag{3.2}$$

where $G(\omega)$ is a high-level task operating on an element of the input sample space Ω. Let the random variable in (3.1) be explicitly defined such that $\mathcal{X} : \Omega \rightarrow \mathbb{R}$ describes the OD system's input. The input is contained within a base communication channel. For example, if the OD system's input are specific points in a LiDAR point cloud, then the base communication channel contains all points in a LiDAR point cloud. Let $\mathcal{X}_0 : \Omega_0 \rightarrow \mathbb{R}$ denote the random variable describing the base system's communication channel. An important distinction is the OD system's input is dependent on the base system's communication channel such that the information in the OD input is never greater than the information in the base channel. This implies $\Omega \in \Omega_0$. By assuming the base system communication channel is the only means by which an input comes into the system,

$$H(\mathcal{X}) \leq H(\mathcal{X}_0), \tag{3.3}$$

where $H(\mathcal{X})$ and $H(\mathcal{X}_0)$ are the information associated with $\mathcal{X}$ and $\mathcal{X}_0$, respectively. Let $H^*(\mathcal{X})$ denote the minimum information such that a figure of merit or evaluation criteria $F(\mathcal{Z})$ does not go below a threshold b. Since $\Omega \in \Omega_0$, $\mathcal{X}$ can be obtained by removing elements from Ω_0 until $F(\mathcal{Z}) = b$. The figure of merit is a constraint that ensures (3.2) remains valid. Unfortunately, a method for directly calculating $H^*(\mathcal{X})$ does not exist. By assuming $H^*(\mathcal{X})$ is directly proportional to how well the task $G(\omega)$ is executed,

$$H^*(\mathcal{X}) = \min(H(\mathcal{X}_0)) \text{ w.r.t. } \Omega_0 \text{ s.t. } F(\mathcal{Z}) \geq b. \tag{3.4}$$

Equation (3.4) implies that finding the minimum information for an input consists of removing possible inputs until the task can no longer be performed to a desired specification. The information contained in the remaining available inputs is the quantity shown in (3.4). In the obstacle avoidance case, if the CNN's figure of merit $F(\mathcal{Z})$ remains at b after using the importance map, then the importance map is not discarding relevant information. The

figure of merit in this application is training loss.

3.1.2 Convolutional Neural Networks

Convolutional neural networks started to become popular in the 2010s when networks such as AlexNet [29] and VGG-16 [30] were achieving stunning results on the ImageNet dataset. This naturally brought CNNs to the autonomous driving scene. Instead of traditional fully-connected neural networks where the forward pass or input data processing is a series of matrix multiplications, CNNs have layers where the forward pass is a convolution operation [31]. CNNs are usually used on images or spatial data because the network learns filters, or kernels, that find key features in the data. These filters find spatial features in the input data that are conducive to the CNN producing the desired output. An example of these features is shown in Figure 3.1, where VGG-16 is retrained to distinguish between carp species [32]. Notice these features begin as simple lines and then begin to reflect higher-level characteristics of the carp pictures as the layer number increases. Despite this example not relating to autonomous driving, the capability to learn spatial features is characteristic of all CNNs, regardless of application.

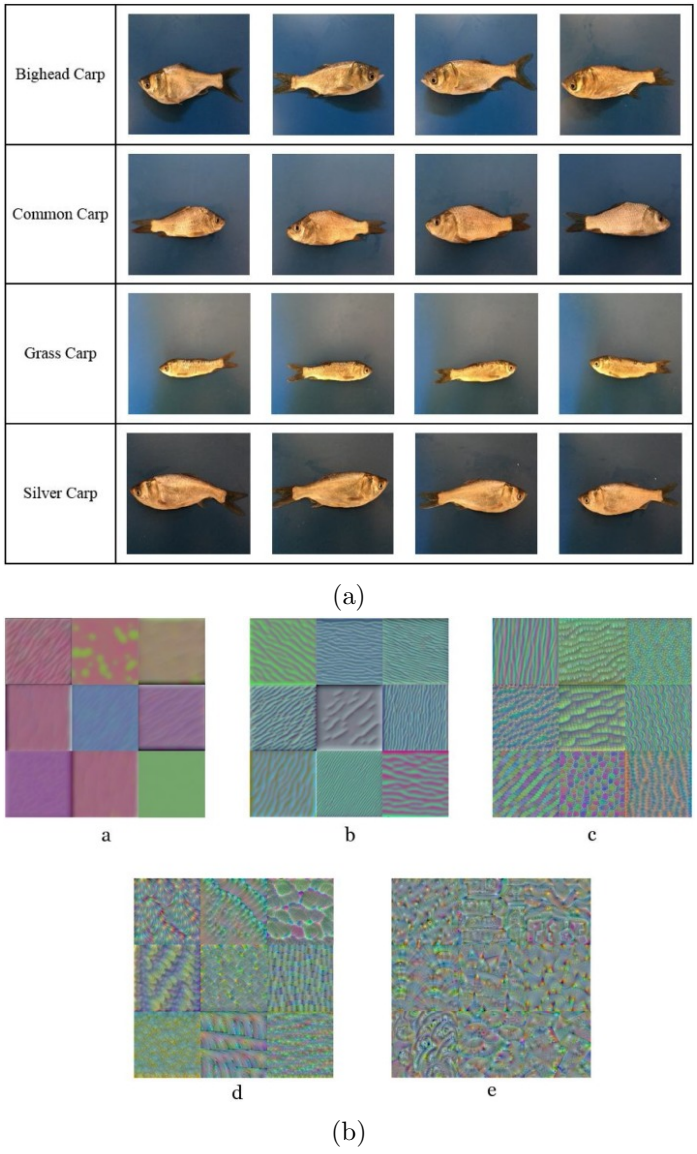


Fig. 3.1: Images show the input data and filters for identifying carp species. (a) Example input images. (b) Filters taken from consecutive convolutional layers in the CNN.

Since CNNs use filters, they can be much more efficient than using a fully-connected network, where each input pixel corresponds to a weighted neuron. Fully-connected networks and CNNs are trained using an algorithm called back-propagation, which was first introduced by Rumelhart et. al. [33] This method uses the chain rule for taking derivatives to utilize previously calculated sensitivities, or the changes in total cost with respect to the neuron weights, to calculate sensitivities deeper in the network until arriving at the input.

Wilmott offers a contemporary explanation of back-propagation in Section 10.11 of [34].

Training a CNN was selected as the mechanism for verifying the importance map because CNNs are loosely based on the human neural system, accessible through popular deep learning programming libraries, and have been used in imitation learning for autonomous driving. Ly and Akhloufi discuss the popularity of estimating control commands directly from sensor data from learning by imitation [35]. A simple architecture that implements an imitation driving neural network is PilotNet [36].

CNN Architecture

The CNN used in this research closely follows the architecture described in [36], which is shown in Figure 3.2.

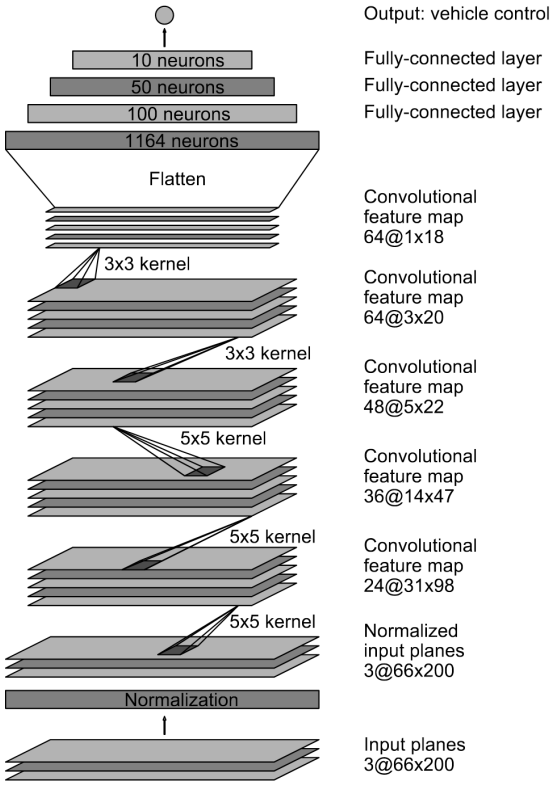


Fig. 3.2: Original PilotNet architecture.

The original PilotNet architecture was slightly altered for the importance map verification:

- Input images were single channel so there was only 1 input plane
- Input images were larger than 66×200 (63×1041)
- The first fully-connected layer has 7872 neurons, according to size calculations [37]
- There are two outputs: one for steering angle and one for vehicle acceleration

The input image size is 63×1041 due to the number of rows and columns in the LiDAR scan. Since the LiDAR scans are stored as unordered point clouds in the available data sets, rounding errors cause the number of rows and columns to be slightly different from the LiDAR specifications. The CNN has no pooling layers. The activation function, or nonlinear transformation on a neuron's output, between the convolution layers is a rectified linear unit (ReLU) function and the activation function between the fully-connected layers is a hyperbolic tangent. ReLUs are not used in the fully-connected layers because the output of the network can be negative and the ReLU activation function has the range $[0, \infty)$.

3.2 Experiment

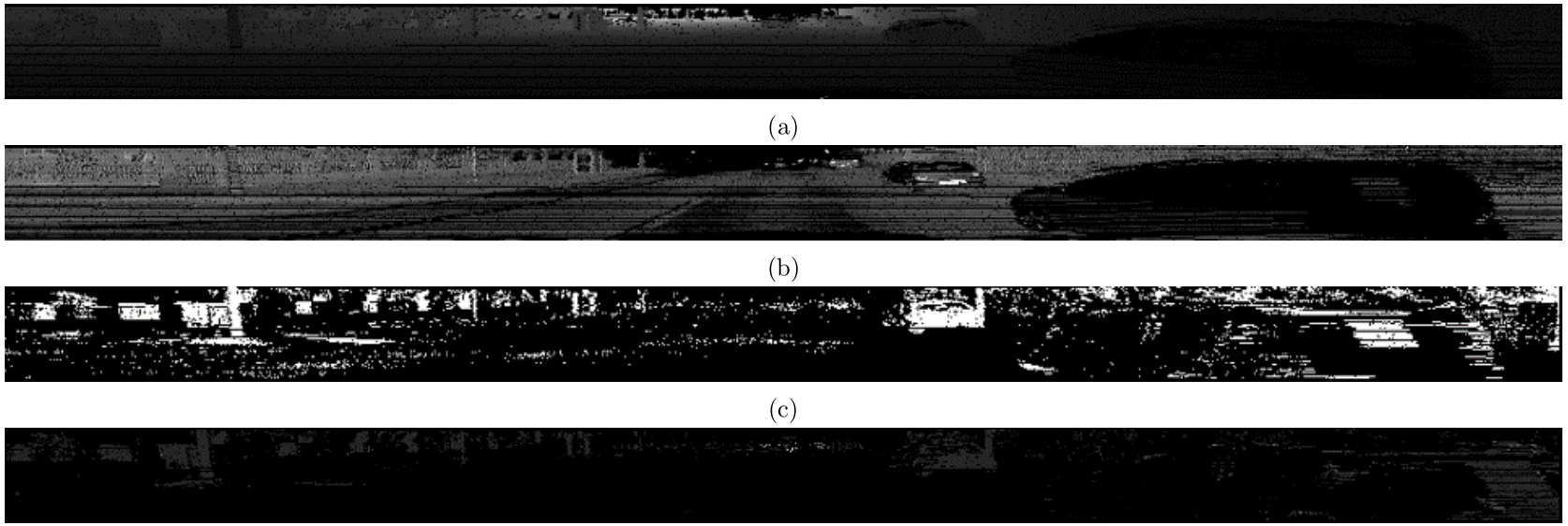
Imitation learning was used to verify the importance map. Even though the presented CNN architecture provides a means for avoiding obstacles in an autonomous vehicle, the purpose of this experiment is to verify no essential information is being discarded when using the importance map for obstacle detection. It is not the purpose of this experiment to create a fully-functioning obstacle avoidance system. Training data was created from time-synchronized data in the KITTI dataset [27]. The KITTI dataset is recorded data from real driving situations. The trained network should produce the same vehicle control commands as a human when given an image of the same scene the human is seeing. Vehicle longitudinal acceleration was directly recorded from an IMU in the KITTI dataset, but steering angle was not. The steering angle was calculated using the Ackermann vehicle model

$$\omega = \frac{v}{l}\tan(\psi), \tag{3.5}$$

where ω is the yaw rate of the vehicle, v is the longitudinal vehicle velocity, l is the distance between the front and rear axle, and ψ is the steering angle [38]. Measurements and images with the same time stamp were not used together. A human has a reaction time around 0.1 seconds and vehicle control systems also have a time constant or delay. Therefore, the vehicle measurements that result from a vehicle command do not occur until after a scene is presented to the driver. To ensure the CNN was learning the correct commands, an image and the vehicle measurements that occur 0.2 seconds after the image were treated as a single training data sample and label. The CNN was trained on the measured acceleration value divided by 10 and the steering angle expressed radians. This places the output of the CNN in the approximate range of $[-1, 1]$ with some of the values being on the order of 0.01. With all the values being in a relatively small range, the training loss (mean-square error or MSE) must be on the order of about 1e-5 for the network to actually be tracking the correct output. A decrease in training loss, or training error, indicates an increase in the CNN's ability to avoid obstacles that exist within the training dataset.

The KITTI data was converted from its native format to ROS bags. ROS is a popular, open-source framework for implementing robotic systems [39]. The LiDAR data was played back and processed to create 4 types of 1-channel images: range image, intensity image, event map, and importance map. Figure 3.3 shows examples of all 4 images. The range image is a scaled and quantized ordered point cloud such that the farthest range is an integer 255 and the nearest range is an integer 0. The intensity image is created from the infrared light intensity field in the point cloud. The event map and importance map are created as described in Chapter 2. These images are all created by converting azimuth and elevation locations of LiDAR returns to column and row coordinates, respectively. This explicit conversion can cause rounding errors, which explains the few black lines across the images in Figure 3.3. None of the images were created with a typical, visible-light camera.

The CNN is implemented and trained using PyTorch, a popular deep learning library



(a)

(b)

(c)

(d)

Fig. 3.3: Figures showing the four different input image types. This scene is from drive 0084. (a) Range image. (b) Intensity image. (c) Event map. (d) Importance map.

for the Python programming language [40]. Tensorboard, a machine learning visualization tool developed originally for TensorFlow, was used to create the plots in Section 3.2.1, Section 3.2.2, and Section 3.2.3. For weight adjustment, an Adam optimizer with a learning rate of 1e-3 is used along with a learning rate scheduler [41]. The learning rate scheduler decreases the learning rate of the optimizer when CNN improvements become minimal. Each network implementation is trained using a batch size of 1. This means the network's weights are updated with back-propagation and an optimizer after executing a forward pass with every image in the dataset. Training through the entire dataset once constitutes an epoch. Each network implementation was trained for 50 epochs. Ten implementations were created for each image type.

Three datasets were created from scenes in three different data recording sessions in the KITTI dataset. These datasets mostly capture moments when obstacle avoidance maneuvers are occurring. Some examples of this are the ego vehicle stopping for a vehicle in traffic, avoiding and stopping for pedestrians, avoiding oncoming vehicles, etc. The datasets are not simply recordings of the ego vehicle driving on a road because learning to drive and following traffic rules is different from learning to avoid obstacles.

3.2.1 Drive 0084

This dataset is created from the last 10 seconds of recorded data in the file 2011_09_26_drive_0084 from the KITTI dataset. There are 108 frames. The ego vehicle is stopping for another vehicle that is stopped at a traffic light. Figures 3.4 - 3.7 show the average training loss for the range image, intensity image, event map, and importance map, respectively. Most of all the CNN implementations using this dataset converge to a training loss around 1e-3. However a few implementations reach a loss on the order of 1e-5. Loss on this order will be referred as "low" training loss. Table 3.1 shows when the CNN is trained using the event map or importance map, the CNN is more likely to achieve a low training loss.

Figure 3.4 shows most of the implementations not improving after the first few epochs. However, there are three implementations that continue to learn by decreasing the training loss. This implies the three implementations with low training loss learned how to avoid

obstacles in the 0084 dataset. When training neural networks, it is very typical to see some implementations train extremely well and other implementations train poorly. With each implementation, the neural network weights are initialized to different values using a Kaiming uniform distribution, or He initialization [42, 43].

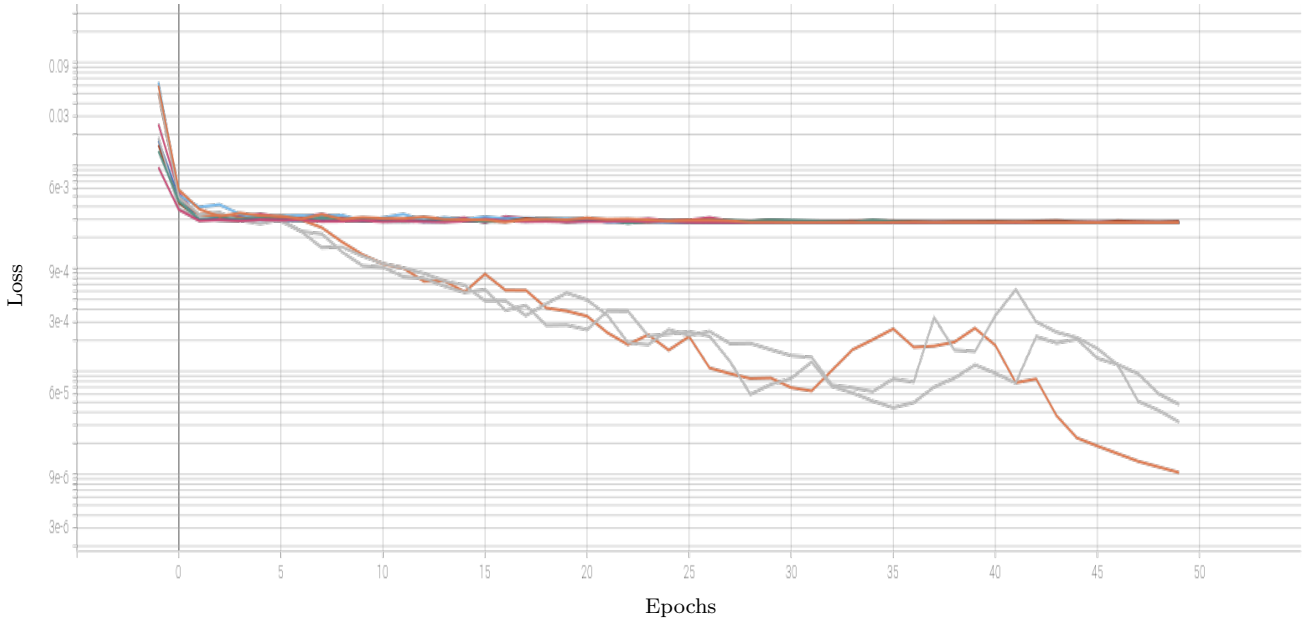


Fig. 3.4: Average training loss on a logarithmic scale with respect to each epoch when training with the range image.

Figure 3.5 is very similar to Figure 3.4 with the main difference being an additional implementation achieved low loss. Due to the strong similarities in training performance, the OA system does not see huge differences between using range or intensity images as input.

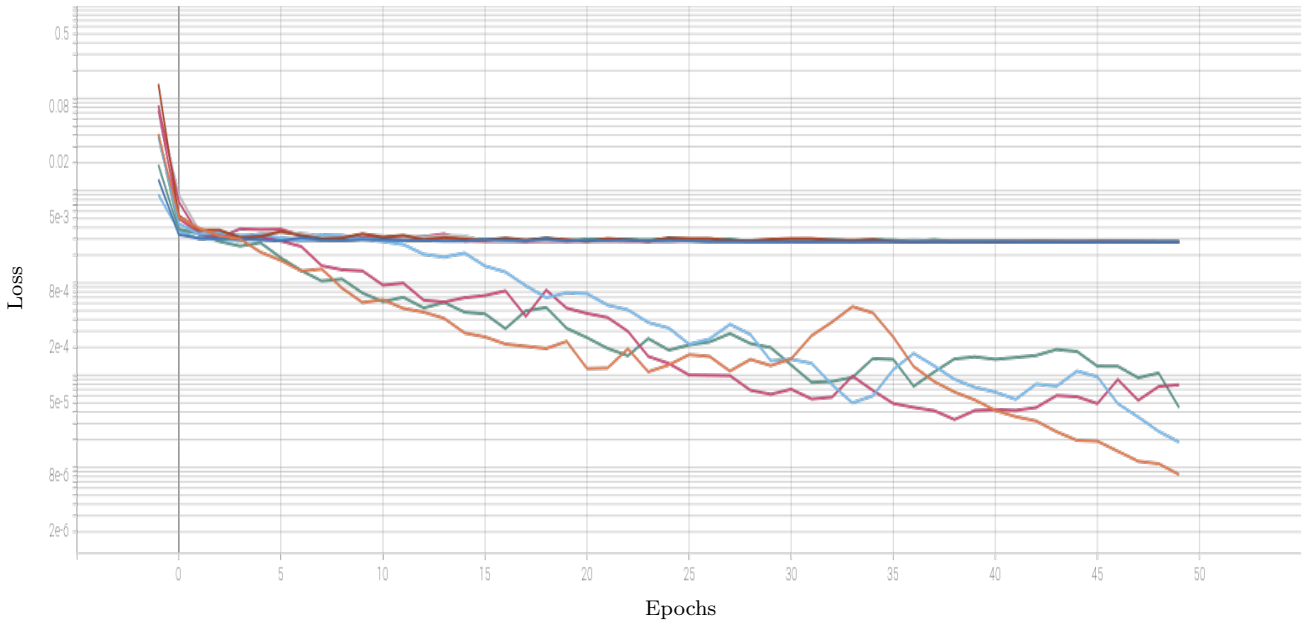


Fig. 3.5: Average training loss on a logarithmic scale with respect to each epoch when training with the intensity image.

From Figures 3.6 and 3.7, there are more implementations that achieve a low training loss than the more typical approaches in Figures 3.4 and 3.5. CNN implementations using an event or importance map have a higher chance of achieving low loss or, in other words, high OA performance than implementations using a range or intensity image.

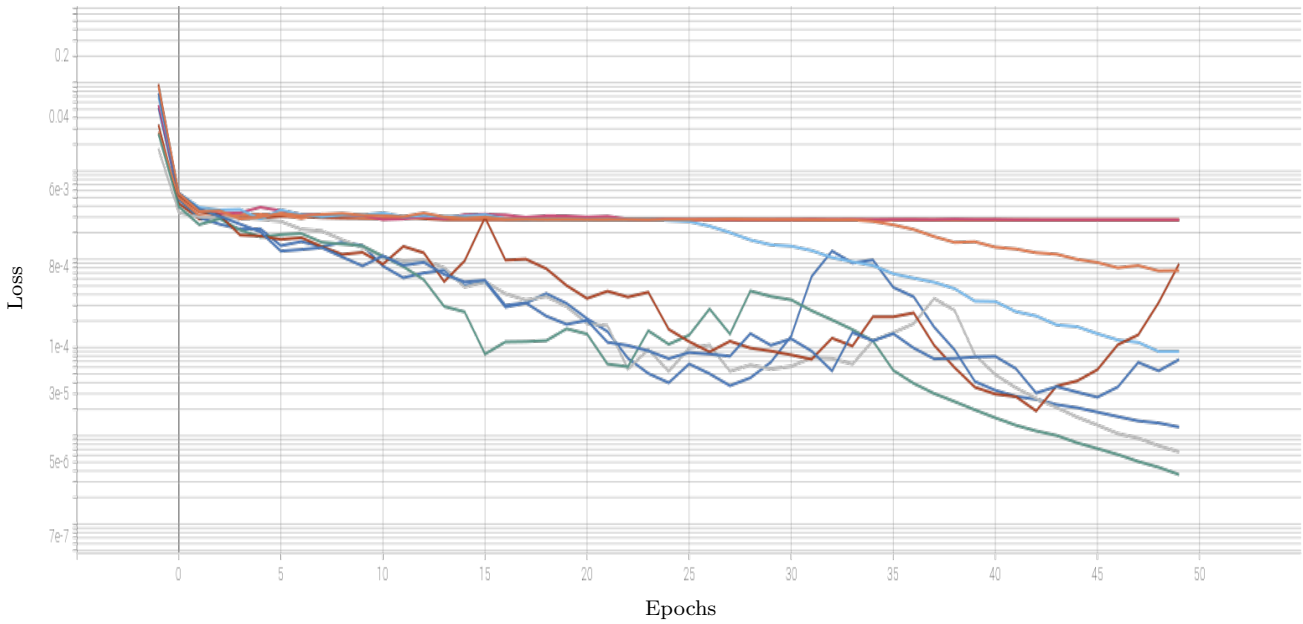


Fig. 3.6: Average training loss on a logarithmic scale with respect to each epoch when training with the event map.

Fig. 3.7: Average training loss on a logarithmic scale with respect to each epoch when training with importance maps from the 0084 dataset.

Table 3.1: Number of CNN Implementations Reaching Low Training Loss

Data Type	Implementations
Range	3
Intensity	4
Event	5
Importance	6

3.2.2 Drive 0022

This dataset is created from two scenes in file 2011_09_26_drive_0084 from the KITTI dataset. There are 87 frames. The first scene occurs around second 56 when an oncoming vehicle turns into the ego-vehicle's traffic lane. The second scene occurs shortly after when

there are two oncoming vehicles in the opposite lane on a thin road. In the second situation, the vehicle moves slightly to the side to give more room to the oncoming vehicles. Figures 3.8 - 3.11 show the average training loss for the range image, intensity image, event map, and importance map, respectively. Due to the complexity of the obstacle avoidance maneuvers, none of the CNN implementations reached a loss on the order of 1e-5. However, the range and intensity images usually corresponded to CNN implementations with a training loss approximately 0.05 higher than the event map and importance map implementations. The sparcity of the new data types probably attributed to obtaining a lower loss. The event and importance map also each had an implementation reach approximately a training loss of 3e-3.

Figures 3.8 and 3.9 are nearly identical except for a single iteration in the range image implementation that does not converge to the typical 0.07, but rather reaches 0.045. The backpropagation training algorithm does not find a combination of weights that reduces the training error to an acceptable value in these two cases.

Fig. 3.8: Average training loss on a logarithmic scale with respect to each epoch when training with range images from the 0022 dataset.

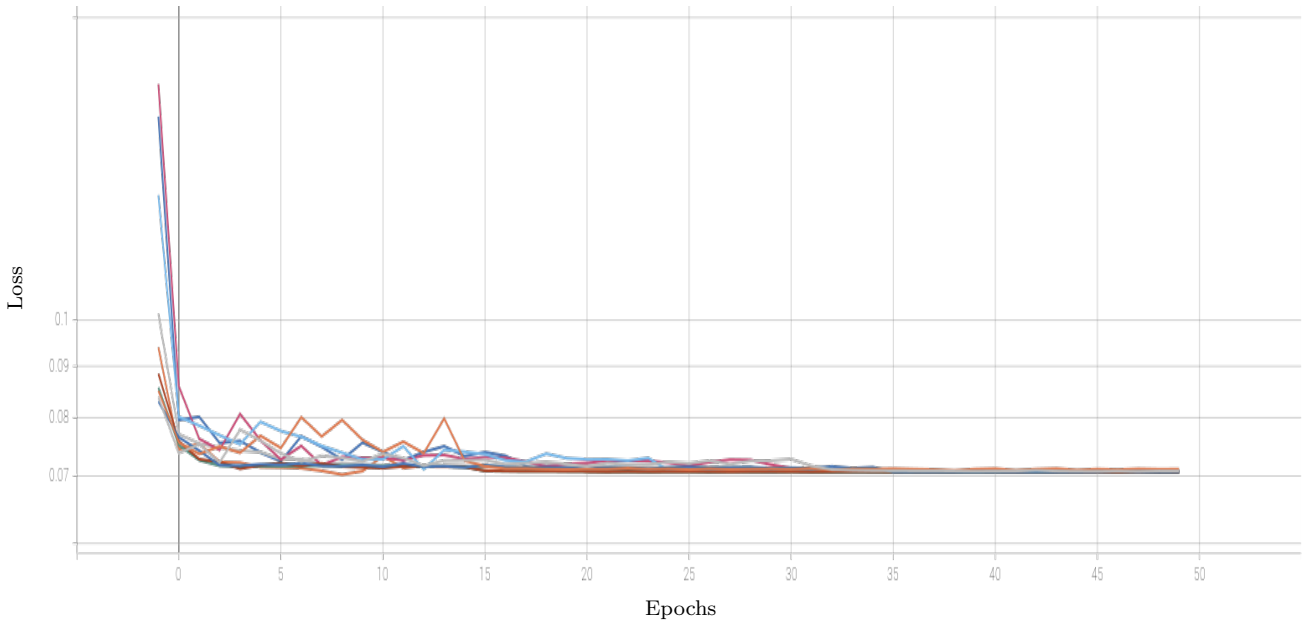


Fig. 3.9: Average training loss on a logarithmic scale with respect to each epoch when training with intensity images from the 0022 dataset.

An interesting observation between Figures 3.8 and 3.9 and Figures 3.10 and 3.11 is the typical loss is 0.02 for the event and importance map implementations. The leading hypothesis for this decrease in loss is the sparcity of these two maps; whereas the range and intensity images have non-zero values at almost every pixel coordinate. Further investigation will be required to determine the true cause of this phenomenon.

The event map and importance map each have a single CNN implementation able to achieve a loss an order of magnitude less than the typical value. Even though the low loss is not present when training against the 0022 dataset, the event and importance map CNN implementations consistently achieve a lower training loss than that of the range and intensity images.

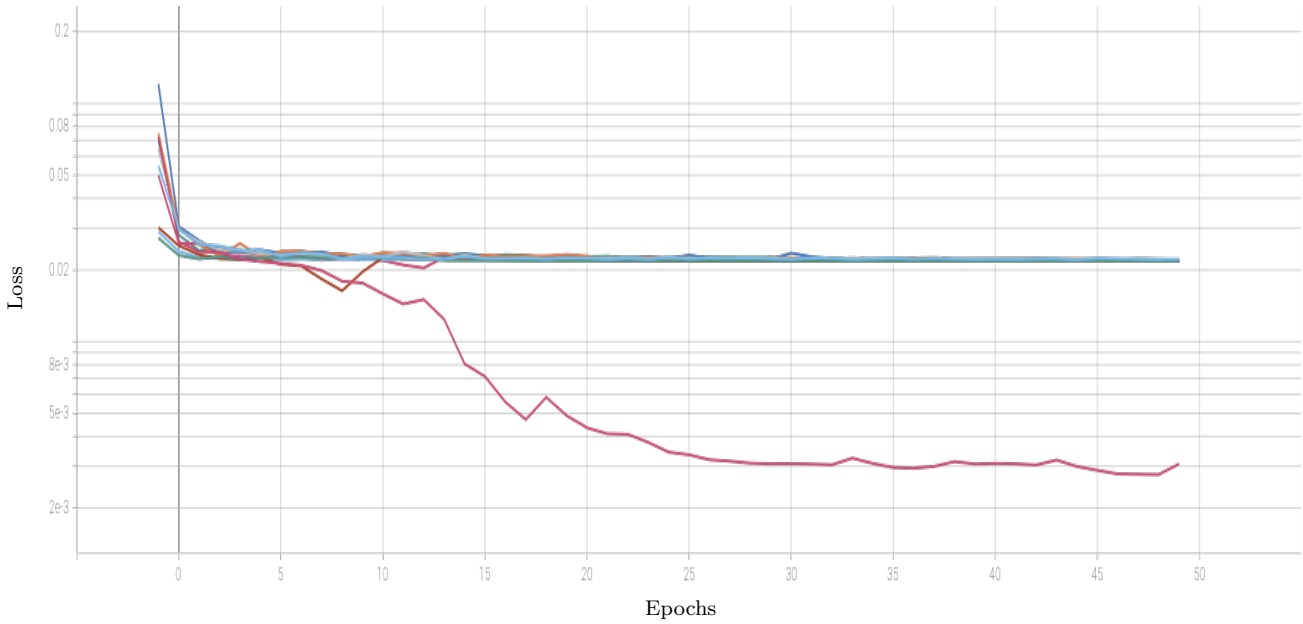


Fig. 3.10: Average training loss on a logarithmic scale with respect to each epoch when training with event maps from the 0022 dataset.

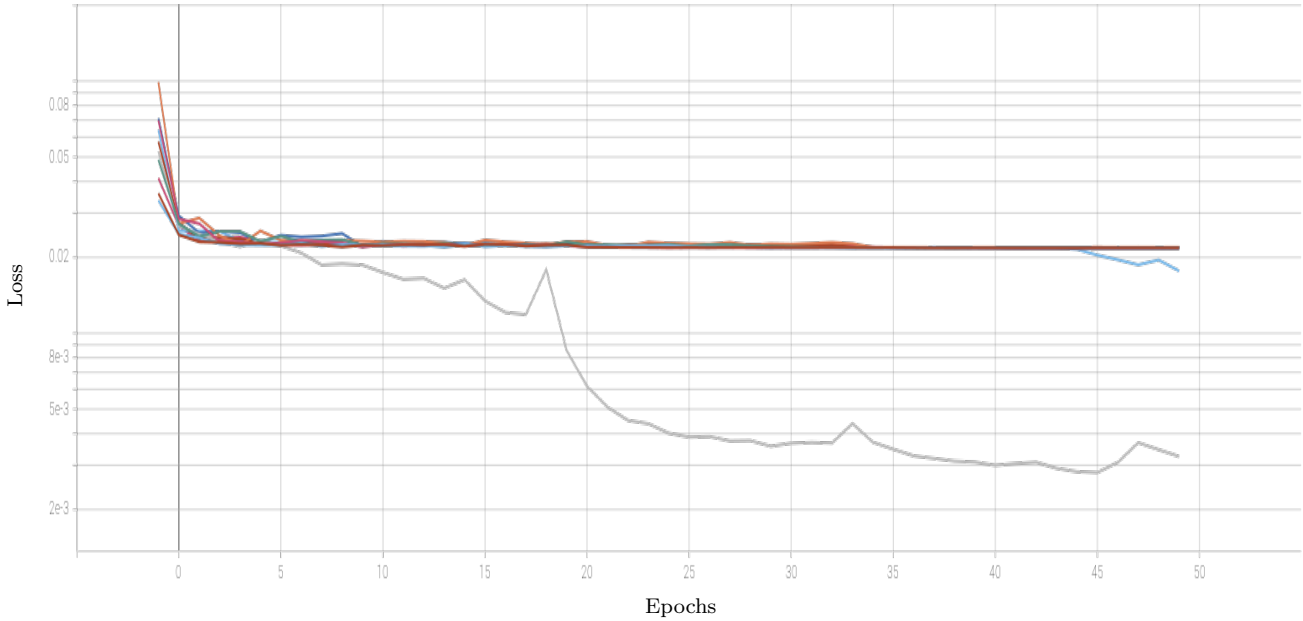


Fig. 3.11: Average training loss on a logarithmic scale with respect to each epoch when training with importance maps from the 0022 dataset.

3.2.3 Drive 0071

This dataset is created from several scenes in file 2011_09_29_drive_0071 from the KITTI dataset. There are 337 frames. The ego vehicle avoids an oncoming vehicle, cyclists, and various groups of pedestrians while driving slowly in an urban environment. Figures 3.12 - 3.15 show the average training loss for the range image, intensity image, event map, and importance map, respectively. There is no significant difference between CNN implementations using different image types. The event and importance map CNN implementations perform just as well as the range and intensity CNN implementations. One range image implementation and one importance map implementation reached a training loss on the order of 1e-5.

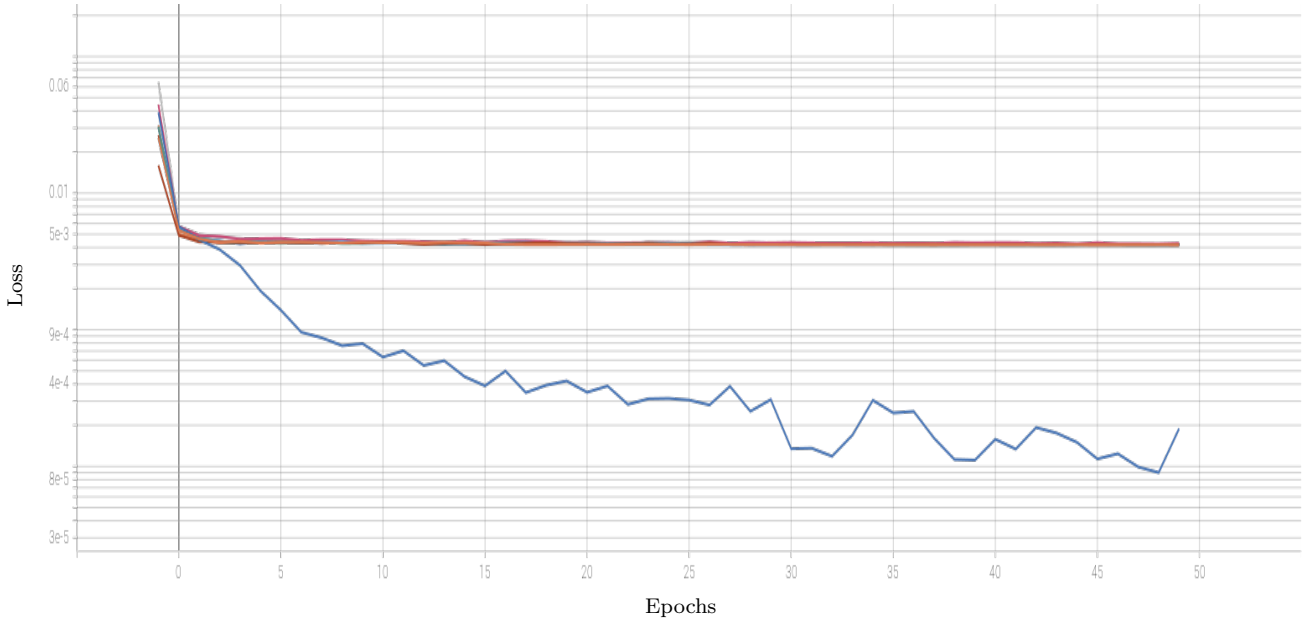


Fig. 3.12: Average training loss on a logarithmic scale with respect to each epoch when training with range images from the 0071 dataset.

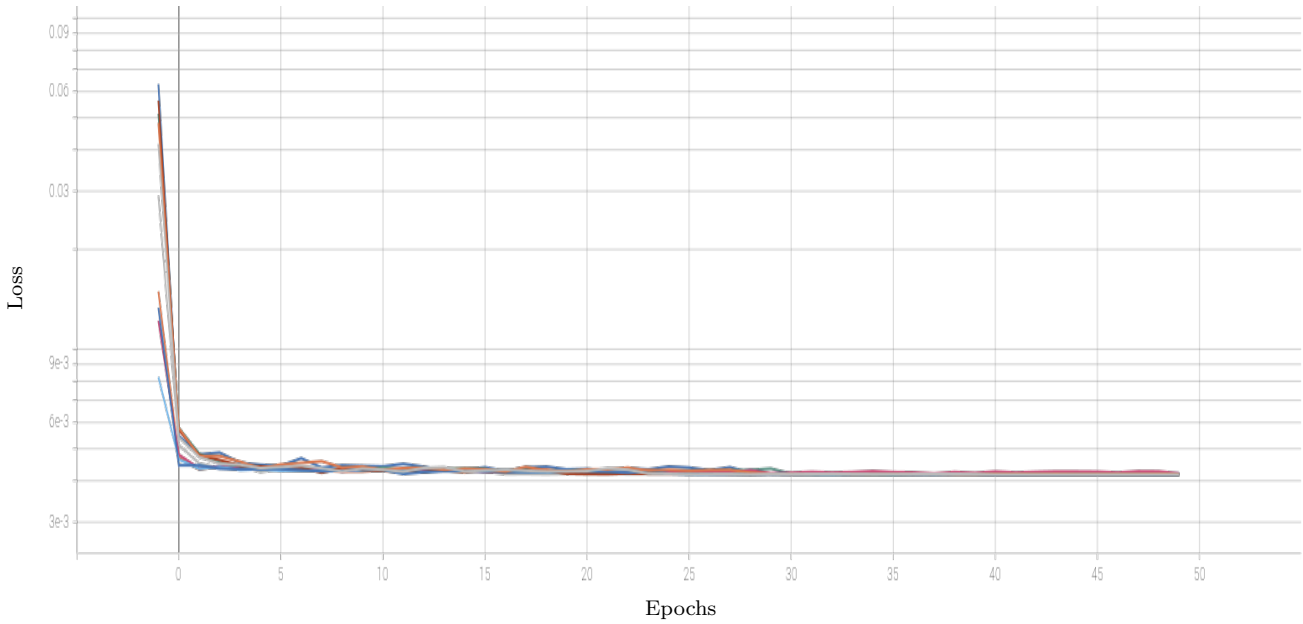


Fig. 3.13: Average training loss on a logarithmic scale with respect to each epoch when training with intensity images from the 0071 dataset.

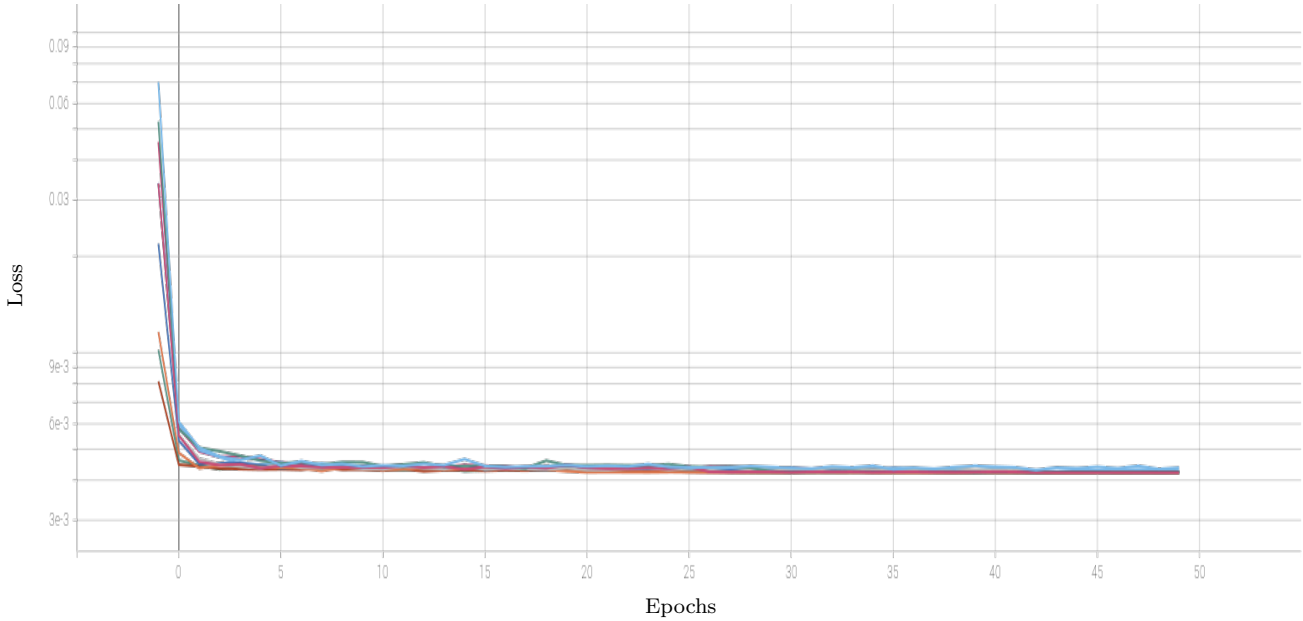


Fig. 3.14: Average training loss on a logarithmic scale with respect to each epoch when training with event maps from the 0071 dataset.

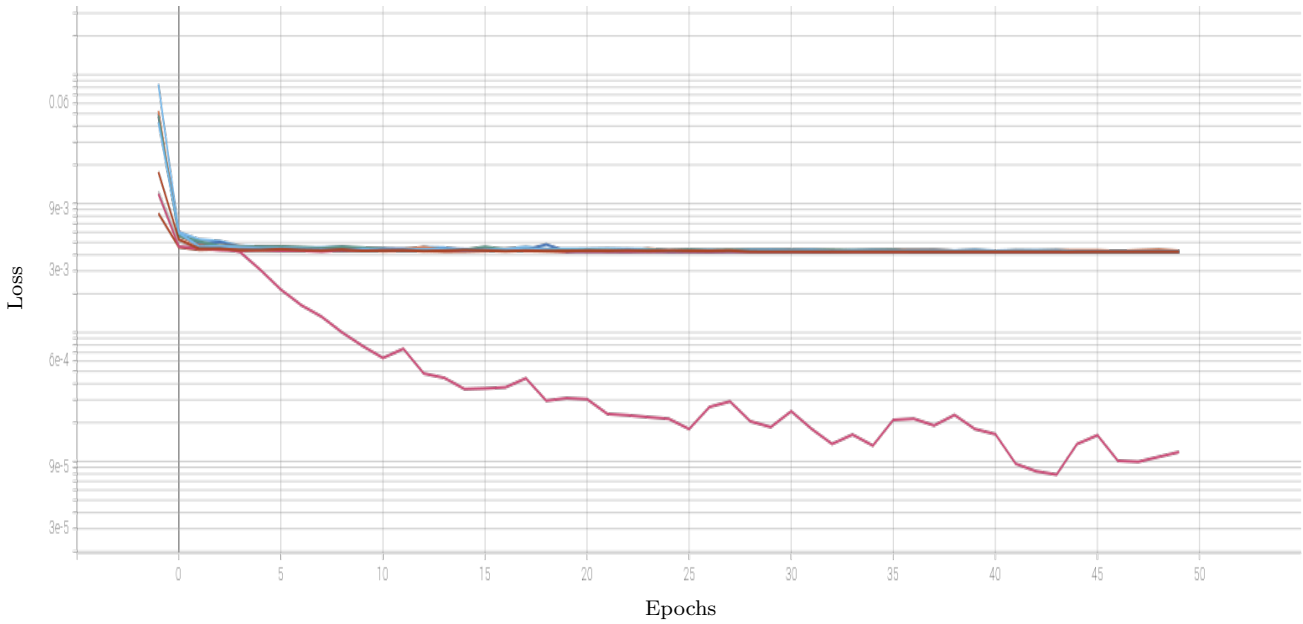


Fig. 3.15: Average training loss on a logarithmic scale with respect to each epoch when training with importance maps from the 0071 dataset.

3.2.4 Comments

From observing the training loss plots in Sections 3.2.1, 3.2.2, and 3.2.3, it is fairly obvious the CNNs are usually overfitted because the training loss does not significantly decrease after the 5th epoch. The epoch count was kept at 50 for two reasons. First, since these networks were only being trained to evaluate the importance map, overfitting was not a problem. Second, sometimes the optimizer would get out of a local minima and continue to decrease the loss, albeit at a much slower rate.

3.3 Conclusion

No information pertinent to obstacle avoidance is lost when creating the event map or importance map because the CNN obstacle avoidance system, when trained with these image types, performs just as well or better than the same obstacle avoidance system trained with more traditional image types. The results in this chapter satisfy objective 2 as stated in Chapter 1. The CNN trained with the event map performed remarkably well considering the event map is a binary image and the other image types contain 255 different quantization

levels per pixel. This knowledge could push path planning engineers and others directly involved with obstacle avoidance to use events to lower network bandwidth or improve current OA systems.

CHAPTER 4

VEHICLE STATE ESTIMATION FOR THE IMPORTANCE MAP

The importance map has been presented as a novel method for detecting obstacles at highway speeds and in dynamic environments. Currently, the importance map is dependent on LiDAR measurements and vehicle state measurements. Vehicle state measurements are usually obtained from inertial navigation systems (INS) that utilize GPS and inertial sensors to estimate the vehicle's position, velocity, and acceleration. An OD system that requires expensive imaging sensors and a complete navigation system is difficult to maintain, install, and productize. To improve hardware simplicity and cost, the vehicle states will be estimated from the LiDAR measurements.

Another objective for vehicle state estimation is to maintain the pixel-wise calcuability of the importance as much as possible. Pixel-wise calculations not only reduce how many times each pixel must be processed but also helps preserve the event-based paradigm of the importance map. For this reason, there will be no spatial feature calculations to find correspondences between consecutive LiDAR scans.

4.1 Theory

The importance map requires only two vehicle states: angular velocity ω and longitudinal velocity v. The vehicle state estimation system has three principle components:

- Measurement Proposals
- Measurement Calculation
- State Estimation

State estimates are propagated from the last time step using the state estimation component. Measurement proposals are created by perturbing the propagated states estimates. Note the measurement proposals are not actual measurements: they are the system's best

guess for the actual measurements. Using these proposals, the measurements are calculated. The state estimation component then uses the calculated measurements to update states within the state estimation component. The desired state estimates, $\hat{\omega}$ and $\hat{v}$, are then obtained directly or indirectly from the interior states of the state estimation component.

In this vehicle state estimation system, measurement proposals consist of values sampled near the propagated state. Point registration using a weighted, singular-value decomposition (SVD) method calculates measurements. An extended Kalman filter (EKF) is the state estimation component. These three components are described in subsequent sections.

4.1.1 Measurement Proposals

This step entails selecting the data to be used during measurement calculation. The previous state estimates are propagated from the previous time step to the current step:

$$(\hat{\omega}_{k-1}^{+}, \hat{v}_{k-1}^{+}) \rightarrow (\hat{\omega}_{k}^{-}, \hat{v}_{k}^{-}), \tag{4.1}$$

where the symbol "+" conveys the state has been updated with a measurement at the specified time step and "−" conveys the state has not been updated with a measurement at the specified time step. State propagation is the process of using previous state estimates and the mathematical system model to predict what the state estimate will be at the next time step, without the use of measurements. Along with the state estimates, the state covariance matrix Σ is also propagated. The propagated state covariance captures errors in the state estimation component's system model. Theoretically, the updated state should be "close" to the propagated state, assuming the system model is accurate. Measurement proposals are sampled from the Gaussian distribution specified by the propagated state mean and covariance.

Generating vehicle state measurements requires finding pairs of points $(\mathbf{p}_i(k-1), \mathbf{p}_i(k))$ such that a point in the previous LiDAR scan $\mathbf{p}_i(k-1)$ is on the same location in the world frame as a point in the current LiDAR scan $\mathbf{p}_i(k)$. These pairs of points will be referred to as point correspondences. Point correspondences must be on objects static in the world

frame, or the measurements will be relative to moving objects instead of the world. An example is a vehicle driver estimating the vehicle's velocity. The driver does not look at vehicles driving in other lanes. If moving vehicles are used as reference points, it appears the driver is moving very slowly forwards or backwards. To obtain a better measurement, the driver looks at buildings, landmarks, and other things that do not move. Ground vehicles certainly do not operate in static environments, so a method for removing non-static objects must be implemented.

One way for identifying point correspondences is to use the static object tracking algorithm developed for the importance map. However, this algorithm requires vehicle states. Without further modifications, the system would be calculating the current vehicle states using the current vehicle states, which is circular reasoning. To avoid this problem, potential vehicle states (measurement proposals) are obtained by sampling from the distribution specified by the propagated states' mean and covariance. Static object tracking can now be employed to classify point correspondences as either static or non-static. The measurement proposal subsystem is able to lock on good proposals by only resampling if the previous proposal sample did not find a static point correspondence. Since points close to each other can be highly correlated, points are not processed in order (i.e. row by row), but rather in random order to prevent proposals from staying on erroneous values.

4.1.2 Measurement Calculation

All the point correspondences can be used to find a rotation matrix R and a translation vector $\mathbf{t}$ from the point cloud at time step $k-1$ to the point cloud at k. The goal is to find the rotation and translation that "best" maps the set of points $\{\mathbf{p}_i(k-1)\}$ to the points $\{\mathbf{p}_i(k)\}$. As in all optimization problems, a criteria for optimality must be selected. In this case, the weighted sum of squared errors (L-2 norm squared) is used because it permits well-known optimization techniques such as finding the minimum of a quadratic function. Appendix B.1 provides more detail. This problem can be presented as

$$(\hat{R}(k), \hat{\mathbf{t}}(k)) = \underset{R(k),\mathbf{t}(k)}{\arg\min} \sum_i \alpha_i ||\mathbf{p}_i(k) - R(k)\mathbf{p}_i(k-1) - \mathbf{t}(k)||_2^2, \tag{4.2}$$

such that α_i is a weight in the range $[0, 1]$. Let

$$P(k) = \begin{bmatrix} \bar{\mathbf{p}}_1(k) & \bar{\mathbf{p}}_2(k) & \dots & \bar{\mathbf{p}}_n(k) \end{bmatrix} \tag{4.3}$$

and

$$P_w(k) = \begin{bmatrix} \alpha_1\bar{\mathbf{p}}_1(k) & \alpha_2\bar{\mathbf{p}}_2(k) & \dots & \alpha_n\bar{\mathbf{p}}_n(k) \end{bmatrix}, \tag{4.4}$$

where $\bar{\mathbf{p}}_i(k)$ is normalized with respect to the weighted average of the points $\{\mathbf{p}_i(k)\}$. Let $SVD(P(k-1)P_w(k)^T) = U\Lambda V^T$. Then it can be shown that

$$\hat{R}(k) = VU^T \tag{4.5}$$

and

$$\hat{\mathbf{t}}(k) = \boldsymbol{\mu}_w(k) - \hat{R}(k)\boldsymbol{\mu}_w(k-1), \tag{4.6}$$

where $\boldsymbol{\mu}_w(k)$ is the weighted average of $\{\mathbf{p}_i(k)\}$. Since V and U are orthonormal by definition of the SVD, $\hat{R}(k)$ is also orthonormal and a valid rotation matrix. See Appendix B.1 for a complete derivation.

Finding the Weights

Despite reaching a solution, there still remains the question of the values for the weights $\{\alpha_i\}$. Finding a weight α_i that gives an optimal solution is difficult because of error propagation when finding $\hat{R}(k)$ and $\hat{\mathbf{t}}(k)$. Most approaches assign weights that are constant for all points, based on euclidean distance, use color similarity, or errors from the sensor model [44]. Since the point correspondences in this problem are labeled by a static object classifier, a new weighting approach is used. The static object classifier uses the static

object feature as expressed in Section 2.1.1 and a range test. The features described in Section 2.1.1 only check for the angular position of the static object. To improve the fidelity of the static object classifier, the range of the previous point R_{meas} is also checked against the predicted static object range R_{pred}. Appendix B.2 shows how this value is calculated. Let R_s and R_n be thresholds such that $R_n > R_s$. If the point correspondence passes the angular position check and the range error $|R_{meas} - R_{pred}|$ is less than R_s, the point correspondence is labelled as static. If the point correspondence passes the angular position check and the range error is less than R_n but greater than R_s, the point correspondence is labeled as non-static. The upper threshold R_n helps ensure the point correspondences used in the measurement calculations are reasonable and not caused by noise or scene inconsistencies.

The weights used in (4.2) will be the weights that minimize the variance of the total correspondence error while assuming the error is 0-mean. Correspondence error for a point correspondence i is designated as

$$\mathbf{e}_i(k) = \mathbf{p}_i(k) - R(k)\mathbf{p}_i(k-1) - \mathbf{t}(k), \tag{4.7}$$

where $R(k)$ and $\mathbf{t}(k)$ are the ground truth values, not the estimates. However, there is also registration error, or error only caused be inaccurate estimates $\hat{R}(k)$ and $\hat{\mathbf{t}}(k)$. In a practical situation, there is a combination of correspondence and registration error. When the correspondence error is large, the minimization problem in (4.2) no longer finds values $\hat{R}(k)$ and $\hat{\mathbf{t}}(k)$ that closely resemble the ground truth values. Therefore, the static and non-static classifier must be accurate enough such that the number of static correspondence points is the same or greater than the number of non-static correspondence points.

Let the correspondence error associated with ground truth static point correspondences be designated with the superscript s and the correspondence error associated with ground truth non-static point correspondences be designated with the superscript n. The error will be modeled as zero-mean Gaussian random variables such that $\mathbf{e}_l^s \sim N(\mathbf{0}, \Sigma_s)$ and $\mathbf{e}_m^n \sim N(\mathbf{0}, \Sigma_n)$. According to the central limit theorem, Gaussian random variables are appropriate in this situation due to the summation of many sources of error. These sources

include discretization error in the static object tracking, laser beam divergence, and the non-uniform physical structure of objects in the world. The covariance matrices of these two Gaussian random variables are

$$\Sigma_s = \sigma_s^2 I \tag{4.8}$$

and

$$\Sigma_n = \sigma_n^2 I, \tag{4.9}$$

respectively.

We will also assume $\mathbf{e}_l^s \perp \mathbf{e}_m^n$, which implies these two random variables are independent of each other since they are Gaussian-distributed. Also note $\{\mathbf{e}_l^s\} \cup \{\mathbf{e}_m^n\} = \{\mathbf{e}_i\}$ and $\{\mathbf{e}_l^s\} \cap \{\mathbf{e}_m^n\} = \emptyset$. The total correspondence error, without weights, is represented as

$$\mathbf{g} = \sum_{i=1}^{N} \mathbf{e}_i = \sum_{l=1}^{L} \mathbf{e}_l^s + \sum_{m=1}^{M} \mathbf{e}_m^n \implies \mathbf{g} \sim N(\mathbf{0}, L\Sigma_s + M\Sigma_n), \tag{4.10}$$

where L is the number of static point correspondences and M is the number of non-static point correspondences. Notice the time index k has been dropped since all values in (4.10) are obtained at the same time index. Weights will now be introduced into (4.10) to optimally combine the errors such that the variance of $\mathbf{g}$ is minimized:

$$\mathbf{g} = a \sum_{l=1}^{L} \mathbf{e}_l^s + b \sum_{m=1}^{M} \mathbf{e}_m^n. \tag{4.11}$$

The weights a and b are greater than 0 and sum to 1. These constraints are important because the trivial solution ($a = b = 0$) does not allow $\hat{R}(k)$ and $\hat{\mathbf{t}}(k)$ to be found. Furthermore, if the weights do not sum to one, biases are introduced. Since we are using a classifier to determine which point correspondences are static and non-static, there is an associated false-positive rate γ and false-negative rate ζ. False positives are classifications where the point correspondences are non-static but are labeled as static, and false negatives

are classifications where the point correspondences are static but are labeled as non-static. The total correspondence error can now be expressed as:

$$\mathbf{g} = a\left[\sum_{l=1}^{\lfloor L-\zeta L\rfloor} \mathbf{e}_l^s + \sum_{m=\lceil M-\gamma M\rceil}^{M} \mathbf{e}_m^n\right] + b\left[\sum_{m=1}^{\lfloor M-\gamma M\rfloor} \mathbf{e}_m^n + \sum_{l=\lceil L-\zeta L\rceil}^{L} \mathbf{e}_l^s\right]. \tag{4.12}$$

Since the values within the brackets in (4.12) are summations of Gaussian random variables, (4.12) can be simplified to

$$\mathbf{g} = a\mathbf{c} + b\mathbf{d}, \tag{4.13}$$

where $\mathbf{c} \sim N(0, \Sigma_c)$ and $\mathbf{d} \sim N(0, \Sigma_d)$ such that

$$\Sigma_c = (L - \zeta L)\Sigma_s + \gamma M \Sigma_n \tag{4.14}$$

and

$$\Sigma_d = (M - \gamma M)\Sigma_n + \zeta L \Sigma_s. \tag{4.15}$$

Σ_s and Σ_d are scaled identity matrices, Σ_c and Σ_d are also scaled identity matrices. The scalar representations of (4.14) and (4.15) are

$$\sigma_c^2 = (L - \zeta L)\sigma_s^2 + \gamma M \sigma_n^2 \tag{4.16}$$

and

$$\sigma_d^2 = (M - \gamma M)\sigma_n^2 + \zeta L \sigma_s^2, \tag{4.17}$$

respectively. Equation 4.13 can now be viewed as a measurement fusion problem. Two measurements (realizations of $\mathbf{c}$ and $\mathbf{d}$ in this case) are given and the goal is to optimally combine the measurements using a linear combination. This a well-known problem that can be solved using conditional Gaussian densities, maximum likelihood, or least squares [45].

The solution is

$$a = \frac{\sigma_d^2}{\sigma_c^2 + \sigma_d^2} \tag{4.18}$$

and

$$b = \frac{\sigma_c^2}{\sigma_c^2 + \sigma_d^2}. \tag{4.19}$$

In the conditional Gaussian solution, the results in (4.18) and (4.19) are reached recursively by finding $\mathbf{g} = f_{\mathbf{c}|\mathbf{d}}(\mathbf{c}|\mathbf{d})$ where $f(\cdot)$ designates a probability density function. The weights a and b correspond to weights such that the linear combination shown in (4.13) is the conditional density $f_{\mathbf{c}|\mathbf{d}}(\mathbf{c}|\mathbf{d})$.

Throughout this derivation for finding $\{\alpha_i\}$, the weights have been selected according to the sum of correspondence errors and not the sum of squared correspondence errors, as is presented in (4.2). The squared error, or L-2 squared in this case, also causes the weights a and b to be squared. When formulating the initial registration problem as shown in (4.2), each normed correspondence error labeled as static is weighted by a^2 and each normed correspondence error labeled as non-static is weighted by b^2.

Creating Measurements From Point Registration Results

The measurement of the vehicle's longitudinal velocity $v'(k)$ is represented as the euclidean norm, or L-2 norm, of $\hat{\mathbf{t}}(k)$ divided by the time step:

$$v'(k) = \frac{||\hat{\mathbf{t}}(k)||_2}{\Delta t}. \tag{4.20}$$

This is a reasonable extension because it is the average longitudinal velocity during a time step. The measurement of the vehicle's angular rate $\omega'(k)$ is obtained from the rotation matrix $\hat{R}(k)$. Since $\hat{R}(k)$ is unitary, it must be of the form

$$\hat{R}(k) = \begin{bmatrix} \cos(\theta(k)) & \sin(\theta(k)) \\ -\sin(\theta(k)) & \cos(\theta(k)) \end{bmatrix}, \tag{4.21}$$

where $\theta(k)$ is the estimated angular displacement of the ego vehicle from $k-1$ to k. Therefore,

$$\omega'(k) = \frac{\sin^{-1}(\hat{R}(k)_{(0,1)})}{\Delta t}. \tag{4.22}$$

Notice the sine term is used to extract the angle $\theta(k)$. In the domain $[-\pi/2, \pi/2]$, cosine loses the sign information of $\theta(k)$ since cosine is symmetric about 0. The measurements $v'(k)$ and $\omega'(k)$ are only used to update the state if they are within 3 standard deviations of the propagated values $\hat{\omega}_k^-$ and $\hat{v}_k^-$, respectively. Three standard deviations is selected because approximately 99.7% of Gaussian-distributed values lie within 3σ of the mean.

4.1.3 State Estimation

An EKF, an extension of the Kalman filter, is a probabilistic filter that assumes Gaussian-distributed quantities, uncorrelated measurement noise, means of conditional probability densities as optimal, and first order Markovity (the previous state provides enough information to represent the entire past) to simplify otherwise complex and computationally intensive estimation problems [45, 46]. The Kalman filter has been extensively researched and developed since the 1960s. The EKF was selected for this research because the EKF is the best-practice for navigation according to NASA [47]. A typical Kalman filter could not be used in this situation due to the non-linearities present in the Ackermann vehicle model. The measurements $v'(k)$ and $\omega'(k)$ are used to update the states in the EKF after state propagation. Table 4.1 describes the states of the EKF. Note the yaw rate ω is not directly estimated by the EKF. To obtain the yaw rate, we use the Ackermann vehicle model: $\omega = \frac{v}{l}\tan(\psi)$.

Table 4.1: EKF States

Variable	Description
v	Vehicle Longitudinal Velocity
ψ	Steering Angle
a	Vehicle Longitudinal Acceleration
ζ	Change in Steering Angle

Table 4.2 shows the required parameters for the EKF.

Table 4.2: EKF Parameters

Variable	Description
l	Distance between front and rear axle
τ_a	Longitudinal acceleration time constant
τ_ζ	"Change in steering angle" time constant
σ_a	Steady-state acceleration standard deviation
σ_ζ	Steady-state "change in steering angle" standard deviation
σ_v	Velocity measurement standard deviation
σ_ω	Angular rate measurement standard deviation
σ_ψ	Steering angle measurement standard deviation

The state-space representation of the system contains a process and observation equation. These equations are represented as

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \mathbf{p}) + G\boldsymbol{w} \tag{4.23}$$

and

$$\boldsymbol{z} = h(\boldsymbol{x}, \mathbf{p}) + B\boldsymbol{\nu}, \tag{4.24}$$

respectively, where $\boldsymbol{w} \sim N(\mathbf{0}, Q)$, $\boldsymbol{\nu} \sim N(\mathbf{0}, R)$, and $\mathbf{p}$ is the parameter vector. The time dependence of the state vector $\boldsymbol{x}$ is implicit.

Process Model

The process, without noise, is modelled as

$$f(\boldsymbol{x}, \mathbf{p}) = \begin{bmatrix} a \\ \zeta \\ -\frac{1}{\tau_a} a \\ -\frac{1}{\tau_\zeta} \zeta \end{bmatrix}. \tag{4.25}$$

The acceleration state a and change in steering angle ζ are modeled as 1st order Markov processes or exponentially correlated random variables (ECRVs). Process noise is modelled as a white, zero-mean, uncorrelated random process of the form

$$\boldsymbol{w} = \begin{bmatrix} w_a \\ w_\zeta \end{bmatrix} \tag{4.26}$$

with autocorrelation $E[\boldsymbol{w}_{t_i} \boldsymbol{w}_{t_j}] = Q\delta(t_i - t_j)$. Since a and ζ are ECRVs, $q_a = 2\sigma_a^2/\tau_a$ and $q_\zeta = 2\sigma_\zeta^2/\tau_\zeta$ where

$$Q = \begin{bmatrix} q_a & 0 \\ 0 & q_\zeta \end{bmatrix}, \tag{4.27}$$

according to Section 4.11 of [45]. The process noise coupling matrix can be expressed as:

$$G = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{4.28}$$

Observation Model

The observation, without noise, is modelled as

$$h(\boldsymbol{x}, p) = \begin{bmatrix} v \\ \psi \\ \frac{v}{l}\tan(\psi) \end{bmatrix}. \tag{4.29}$$

Notice the Ackermann vehicle model is enforced in the observation model and not in the process model [38]. This significantly simplifies the state model without over-complicating the observation model. Observation noise is modelled as a white, zero-mean, uncorrelated random process of the form

$$\boldsymbol{\nu} = \begin{bmatrix} \nu_v \\ \nu_\psi \\ \nu_\omega \end{bmatrix} \tag{4.30}$$

with autocorrelation $E[\boldsymbol{\nu}_{t_i}\boldsymbol{\nu}_{t_j}] = R\delta(t_i - t_j)$. Section 4.2 will explain how this assumption may be violated in the presented application. The peak autocorrelation value, or variance, is described by

$$R = \begin{bmatrix} \sigma_v^2 & 0 & 0 \\ 0 & \sigma_\psi^2 & 0 \\ 0 & 0 & \sigma_\omega^2 \end{bmatrix}. \tag{4.31}$$

The measurement noise coupling matrix B is simply a 3×3 identity matrix.

State and Covariance Propagation

The state vector $\mathbf{x}$ is propagated using Euler's method for solving ordinary differential equations (ODEs) such that

$$\mathbf{x}_k^- = \mathbf{x}_{k-1}^+ + \Delta t f(\mathbf{x}_{k-1}^+, \mathbf{p}). \tag{4.32}$$

The differential equation showing the change in state covariance is

$$\dot{P} = FP + PF^T + GQG^T, \tag{4.33}$$

where

$$F = \frac{\partial f}{\partial \boldsymbol{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{1}{\tau_a} & 0 \\ 0 & 0 & 0 & -\frac{1}{\tau_\zeta} \end{bmatrix} \tag{4.34}$$

is the Jacobian of f. Notice $f(\boldsymbol{x}, \boldsymbol{p}) = F\boldsymbol{x}$. Since (4.33) is a linear differential equation, using Euler's method for solving is an accurate and practical choice:

$$P_k^- = P_{k-1}^+ + \Delta t \dot{P}_{k-1}^+. \tag{4.35}$$

A typical approach for finding P_k^- is to use the state transition matrix (STM) Φ and linear combinations of Gaussian random variables such that

$$P_k^- = \Phi(k, k-1) P_{k-1}^+ \Phi(k, k-1)^T + S_k, \tag{4.36}$$

where S_k is a discretized version of the process noise covariance matrix Q [47]. The STM is usually estimated with Taylor approximations or Runge-Kutta integrations. Equation 4.35 was selected as the more simple option with minimal error due to the linearity of f and the small propagation time ($\Delta t = 0.1s$).

State and Covariance Update

The linearized observation model is

$$H(\boldsymbol{x}) = \frac{\partial h}{\partial \boldsymbol{x}} \approx \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{l}\tan(\psi) & 0 & 0 & \frac{v}{l}\sec^2(\psi) \end{bmatrix}. \tag{4.37}$$

Upon investigation of (4.37), one will see (4.29) is not a function of ζ, or $\dot{\psi}$. An important assumption in the Ackermann model is that of a constant turning radius or curvature. This is usually a safe assumption in a typical navigation system where only the vehicle heading and position are estimated, but in this application of estimating the steering angle directly, the model fails in some circumstances. For example, according to the Ackermann model,

$$\frac{\partial \omega}{\partial \psi} = \frac{\partial}{\partial \psi}\frac{v}{l}\tan(\psi) = \frac{v}{l}\sec^2\psi. \tag{4.38}$$

Equation 4.38 says as long as the vehicle is travelling forward, any change in steering angle will always increase the yaw rate of the vehicle. This is not accurate. Changing the steering angle to be closer to 0 decreases the yaw rate of the vehicle, when velocity is held constant. The Ackermann model is derived from the characteristics of Ackermann steering geometry and the arc length equation $s = r\theta$ such that s is the distance travelled, r is the turning radius, and θ is the change in the ego vehicle's angular displacement. When the turning radius r is allowed to change, the Ackermann model becomes

$$\omega = \frac{v}{l}\tan(\psi) + \frac{\sec^2(\psi)}{\tan(\psi)}\dot{\psi}\theta, \tag{4.39}$$

where the first term is the contribution from the vehicle velocity and the second term is the contribution from the change in steering angle. Since θ is the change in the angular position in the arc length equation, θ is proportional to the angular velocity ω; therefore, $\theta \approx \frac{v}{l}\tan(\psi)$. This allows for

$$\frac{\partial \omega}{\partial \zeta} \approx \frac{v}{l} \sec^2 \psi, \tag{4.40}$$

which is shown in (4.37). The typical Ackermann model is still used to predict the measurement for simplicity and stability (no possible division by zero).

The residual is expressed as

$$\boldsymbol{r} = \boldsymbol{z}_k - h(\boldsymbol{x}_k^-, \boldsymbol{p}), \tag{4.41}$$

where $\boldsymbol{z}_k$ is the measurement

$$\boldsymbol{z}_k = \begin{bmatrix} v_k' \\ \psi_k' \\ \omega_k' \end{bmatrix}. \tag{4.42}$$

To update the state, the residual is weighted using a quantity known as the Kalman gain

$$K = P_k^- H(\boldsymbol{x}_k^-)^T [H(\boldsymbol{x}_k^-) P_k^- H(\boldsymbol{x}_k^-)^T + BRB^T]^{-1} \tag{4.43}$$

and then added to the propagated state such that

$$\boldsymbol{x}_k^+ = \boldsymbol{x}_k^- + K\boldsymbol{r}. \tag{4.44}$$

The covariance is updated with

$$P_k^+ = [I - KH(\boldsymbol{x}_k^-)] P_k^- \tag{4.45}$$

according to Section 5.6 in Maybeck [45].

4.1.4 System Overview

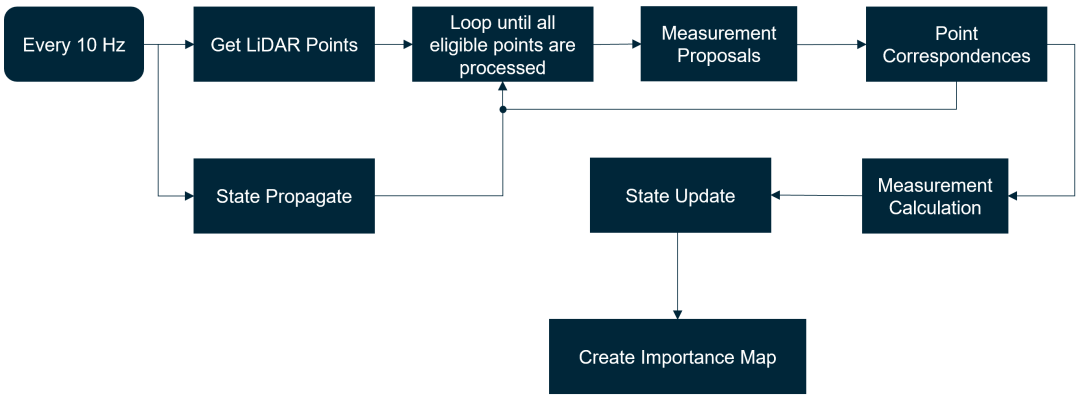


Fig. 4.1: Diagram showing the high-level operation of the vehicle state estimation system.

Figure 4.1 provides an overview of the state estimation system. A LiDAR scan is first received at a rate of 10 Hz. The previous EKF state is propagated to the current time step or, in the case of the initial step, the initial EKF state is used. While looping through the eligible LiDAR points (classified as events and located on the periphery of the LiDAR's field-of-view), new measurement proposals are created at each iteration by sampling from the propagated state's distribution. These proposals are then used in the static object tracking algorithm to find point correspondences. After classifying the point correspondences as either static or non-static and assigning each correspondence an appropriate weight, measurements are calculated using point registration. The calculated measurements $v'(k)$ and $\omega'(k)$ are then used to update the states in the EKF. Using the updated states, the velocity v and yaw rate $\omega = \frac{v}{l}\tan(\psi)$ from the EKF state vector $\boldsymbol{x}$ are used to create the importance map.

4.2 Experiment

Tables in Section 4.2.1 show the average number of misclassified points per frame from the obstacle mask, or the average L-∞ norm per frame. When using the importance map, the error metric is the average L-1 norm per frame, scaled by 1/255. The obstacle mask and

importance map created using the system as described in Chapter 2.1 is the ground truth. Each trial lasted for 45 seconds. The state estimation configuration uses the vehicle state estimation system described in this chapter. The "All Static Object" configuration does not use static object tracking (Section 2.1.1) by labelling all events (Section 2.1.1) as static and uses a constant AHI. This is the same as assuming there are no moving objects in the scene. The "All Non-Static Object" configuration does not use static object tracking by labelling all events as non-static and uses a constant AHI, which is analogous to assuming all objects in the scene are moving. Table 4.3 shows the parameters used in the trials.

Table 4.3: EKF Parameters for Experiments

Parameter	Description	Value
l	wheelbase	2.7m
Δt	time step	0.1s
τ_a	acceleration time constant	2.0s
τ_ζ	change in steering angle time constant	3.0s
σ_a	acceleration std dev	0.8m
σ_ζ	change in steering angle std dev	0.02rad
σ_v	velocity measurement std dev	0.9m
σ_ω	yaw rate measurement std dev	0.05rad
σ_ψ	steering angle measurement std dev	0.8rad

The steering angle measurement ψ'_k is used more as a mechanism for modelling the system than as an actual measurement. If the measurements v'_k and ω'_k are within 3 standard deviations of the respective propagated states and ψ'_k (which is calculated using the Ackermann model with the velocity and yaw rate measurements) is also within 3 standard deviations of the propagated ψ state, the measurement is used in the EKF update equations. If ψ'_k does not lot lie within those bounds, 0 rad is used as a measurement. This effectively models ground vehicles on normal roadways because the steering angle is usually 0, except when travelling on curves or turning.

A point or LiDAR return is only considered for static object classification while finding point correspondences if the point is an event, according to the definition in Chapter 2, and the point has an angular displacement of more than 60 degrees from the vehicle's direction

of travel. Points located on the peripherals are only used because events usually indicate object edges in these situations. When events are mostly object edges, the static object tracking algorithm does not have as many false positives.

The parameters for finding the correspondence weights are listed in Table 4.4.

Table 4.4: Point Registration Weighting Parameters

Parameter	Description	Value
γ	false-positive rate	0.2
ζ	false-negative rate	0.2
σ_s	static point correspondence std dev	0.3m
σ_n	non-static point correspondence std dev	5.0m
R_s	range error for static point correspondence	0.5m
R_n	range error for non-static point correspondence	3.0m

4.2.1 Importance Map and Obstacle Mask Comparisons

Table 4.5 shows results from a KITTI data set in an urban environment with a vehicle driving around a street corner and stopping at a stoplight (2011_09_26_drive_0084). The state estimation configuration has few more obstacle mask errors than the "All Static Object" configuration due to inaccuracies in the vehicle state estimates. Furthermore, assuming all objects in a scene are static is a reasonable option in many circumstances. If this is the case, and the vehicle does not need to see far obstacles on curves, there is no need for static object tracking and that step in the creation of the importance map can be skipped. The state estimation importance map performs better than the "All Static Object" importance map.

Table 4.5: Average Errors for Drive 0084

Configuration	Obstacle Mask Error	Importance Map Error
State Estimation	5.910	33.66
All Static Object	3.454	39.07
All Non-Static Object	1008	1000

Table 4.6 shows results from a KITTI data set in a busy urban environment with

people crossing across the road (2011_09_26_drive_0071). There are many errors in this data set due to the vehicle starting and slowing to a near stop multiple times. This creates many moments where there are either few events or all the events are created by moving objects. Despite the increased error, most of the error is in the form of isolated spots through out the mask with the majority of the ground truth obstacle detection remaining intact with the state estimation obstacle detection. Much of this error can be filtered out using morphological operations.

Table 4.6: Average Errors for Drive 0071

Configuration	Obstacle Mask Error	Importance Map Error
State Estimation	99.00	183.1
All Static Object	70.47	115.7
All Non-Static Object	1175	1254

Table 4.7 shows results from a KITTI data set in a highway environment with a few vehicles merging onto the highway (2011_10_03_drive_0042). The errors between the state estimation configuration and the "All Static Object" configurations are closest in this data set because there is an instance where the static scene assumption is violated.

Table 4.7: Average Errors for Drive 0042

Configuration	Obstacle Mask Error	Importance Map Error
State Estimation	5.044	50.54
All Static Object	4.750	55.04
All Non-Static Object	911.0	873.6

Vehicle state estimation is necessary when there are no vehicle measurements available for the importance map, moving objects are present in the scene, and for seeing obstacles when turning. In Figure 4.2, the ego vehicle is driving on a highway and a vehicle is merging. At the depicted moment, the front of the vehicle is on a collision course with the ego vehicle (See Figure 2.7). Subfigure (a) shows the state estimation configuration is able to detect a moving obstacle whereas the "All Static Object" configuration is not. Figure 4.2 also shows

the obstacle mask using the true vehicle states and the obstacle mask using the estimated vehicle states are very similar.

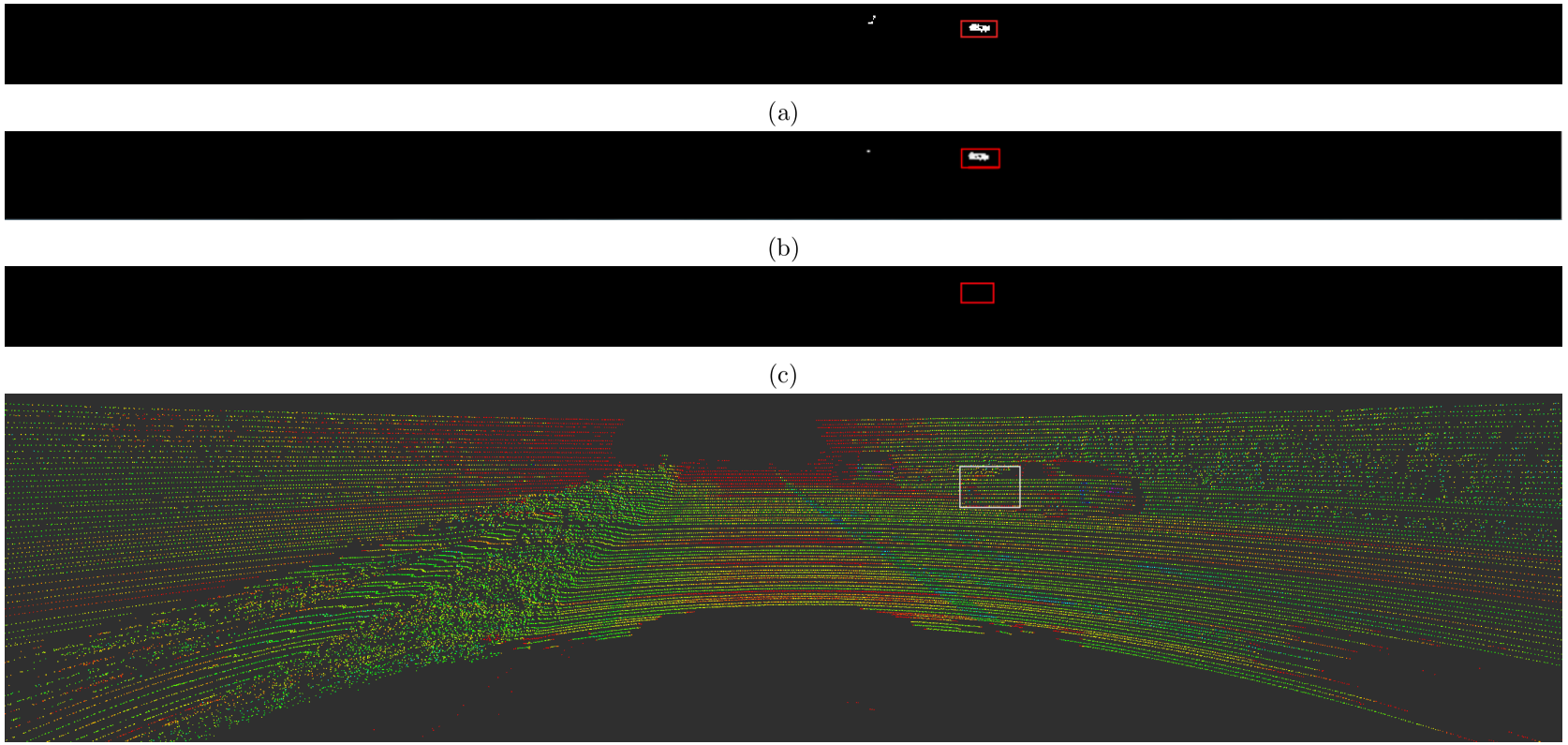
4.2.2 Example Trials

The examples in this section compare truth vehicle states to estimated vehicle states. Since the purpose of the state estimation system is to provide a necessary input for static object tracking and AHI in the importance map, the estimator's requirements are not as stringent as a typical navigation system. An error of 1.0 [m/s] corresponds to an error of approximately 3 pixels in the static object tracking algorithm in the worst case scenario (tracking objects located perpendicular to the vehicle's direction of travel). An error of 0.1 [rad/s] also corresponds to an error of approximately 3 pixels. The importance map has several built-in mitigations to reduce the effects of these errors. During static object tracking the Moore neighborhood of a proposed static object location is checked [20]. The MRF operating on each pixel ensures pixels that are likely obstacles are not immediately labelled as obstacles. Furthermore, static object tracking is the last operation performed in the creation of the importance map. Only about 5% of all LiDAR returns require static object tracking due to the calculations of previous steps that eliminate the majority of returns. As for the calculation of the AHI, the curvature $\frac{\omega}{v}$ is kept at 0 unless the velocity of the vehicles exceeds ± 1.0 [m/s] and the yaw rate of the vehicle exceeds ± 0.1 [rad/s].

Drive 0084

Figures 4.3 and 4.4 show state estimation for drive 0084 (2011_09_26_drive_0084). The velocity and yaw-rate estimation performs fairly well with errors usually below 1.0 [m/s] and 0.1 [rad/s]. Large errors occur near the 0th and 350th time steps because the number of events in the scene is nearly zero.

The oscillations in the yaw-rate estimate occur from a difficulty in modelling the kinematics of the steering angle of a vehicle. When the vehicle is driving in a straight line, the steering angle is a constant. When the vehicle going into a turn, there is a relatively sharp change in steering angle. If the time constant of the change in steering angle τ_{zeta} is



(a)

(b)

(c)

(d)

Fig. 4.2: Figures showing the scene and obstacle masks for drive 0042 at second 19.5. The white and red boxes show the location of an obstacle. (a) The obstacle mask created from the importance map with state estimation. (b) The obstacle mask created from the importance map with true vehicle states. (c) The obstacle mask created from the importance map without state estimation. (d) The scene as a LiDAR point cloud, colored according to intensity.

increased, the oscillations disappear but the system fails to track the yaw-rate when going into a turn and vice-versa.

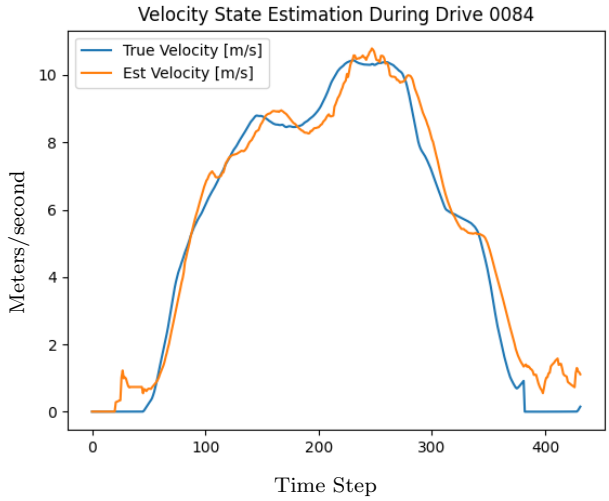


Fig. 4.3: Velocity state estimation during the 0084 data set. The x-axis is time steps and the y-axis is [m/s].

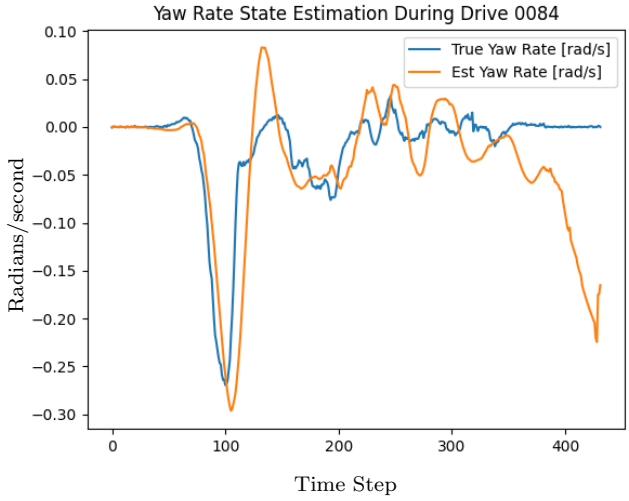


Fig. 4.4: Yaw rate state estimation during the 0084 data set. The x-axis is time steps and the y-axis is [rad/s].

Drive 0042

Figures 4.3 and 4.4 show state estimation for drive 0042 (2011_10_03_drive_0042). The performance of the state estimation system in this data set is interesting because there is a lack of events for the majority of the drive (open highway with little foliage). There is an initial error because this data set starts when the vehicle is currently driving. At the beginning of the driving the velocity and yaw-rate estimates converge to the truth values, then they diverge as the number of events decreases. The estimates converge again at around time step 800.

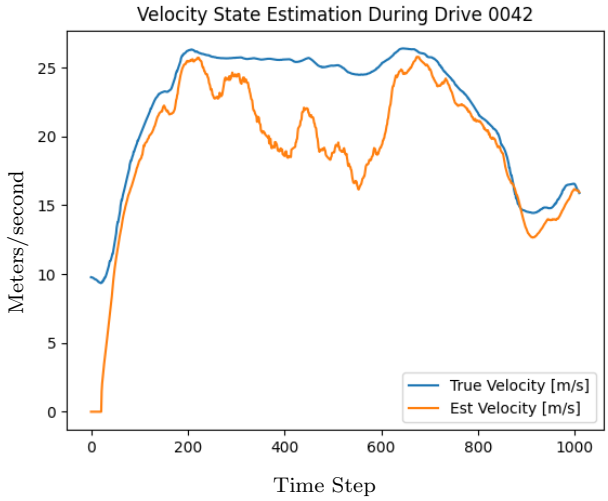


Fig. 4.5: Velocity state estimation during the 0042 data set. The x-axis is time steps and the y-axis is [m/s].

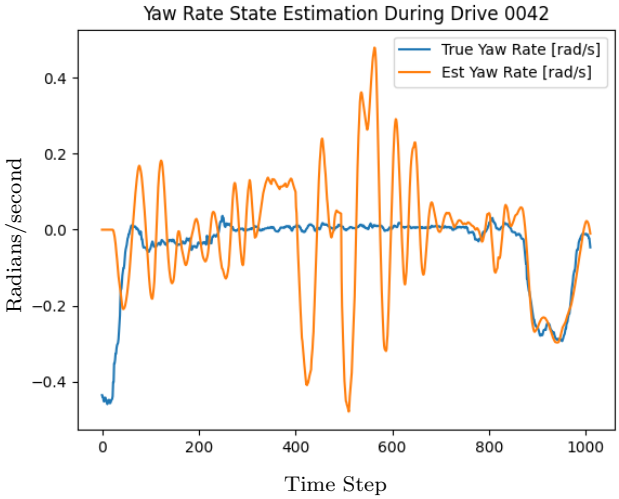


Fig. 4.6: Yaw rate state estimation during the 0042 data set. The x-axis is time steps and the y-axis is [rad/s].

Figures 4.5 and 4.6 show errors that could make the importance map unable to detect moving obstacles outside of the AHI. The cause of this error is driving in an environment with few 3D features. Further research will need to determine how the state estimation system should operate when there are few events or how to use non-event points in these circumstances. However, these figures also show how well the state estimation system recovers from relatively large errors. After the initial error, the system immediately reduces the error and begins to track the truth values. After the period with few events, the system again converges to the truth values. This is a significant finding because velocity and yaw-rate measurements are not independent of the state estimates, which violates the assumption of white observation noise, since estimation error is not white. The fact the EKF is still able to recover from instances where the estimation error is large shows the assumption of white observation noise is not violated enough to derail the entire system.

4.3 Conclusion

The current implementation of the state estimation system performs well when there are sufficient range events in the scene. This system is also able to obtain these estimates

without performing any kernel operations on the image at the current time step and without iterating to find corresponding points. There are also no assumptions about a static environment. These restrictions and lack of assumptions significantly complicate vehicle state estimation. The majority of algorithms that find point correspondences for point registration do not find correspondences in $O(n)$ time; whereas this implementation only needs to use each point once per time step. Furthermore, a unique method for weighting the point correspondences was developed and presented to facilitate this capability. The results in this chapter satisfy objective 3 as stated in Chapter 1. Despite the noted limitations, the presented vehicle state estimation system is a success and viable approach.

Further investigation will be required to improve the state estimation. The quality of the point correspondences in measurement creation can be improved or the model in the EKF can be altered, or perhaps a combination of both. A potential option for the EKF model is to use the vehicle's curvature instead of steering angle.

CHAPTER 5

CONCLUSION

Obstacle detection for autonomous vehicles continues to gain traction as companies try to make their inherently dangerous vehicles safe. With so many academic, commercial, and military entities searching for the silver bullet of autonomous driving in deep learning or other common methods, hopefully this thesis is a refreshing change from typical approaches. The importance map with vehicle state estimation is a stand-alone obstacle detection system that efficiently handles moving and static objects in dynamic environments using inspiration from biology and obstacle scope reduction.

Chapter 2 introduces the importance map and shows the importance map can be used to create an effective obstacle detection system. Another fact shown in Chapter 2 is the utility of event features or the event map as obstacle discriminators. Furthermore, a temporal filter for processing binary signals to enforce a variety of behaviors is presented to identify obstacles in pertinent situations. This filter, the MRF, can also be used independently of the OD system presented in Chapter 2. Combining elements from frame-based processing, neuromorphic systems, and biology enabled the development of a successful OD system. This type of cross-discipline development needs to continue for OD systems to reach the specifications people want on their vehicles. OD is a difficult problem and one of the best ways for creative solutions is to apply ideas in one domain to another. Relying solely on a single discipline will not be sufficient.

Chapter 3 indirectly shows the information content, with respect to obstacle avoidance, is not reduced when using an importance map or an event map. Since the range image contains more data (8-bit pixels) than the event map (1-bit pixels), obstacle avoidance maneuvers can be calculated with much less data. This shows that little temporal information in obstacle avoidance is much more useful than detailed information at a snapshot in time. Engineers can use this fact to make OA more computationally efficient or to

reduce the network bandwidth when sending data across components.

Chapter 4 shows how to make the importance map independent of all sensors other than a LiDAR, making this system more desirable for production teams. This is accomplished by designing a vehicle state estimation system that avoids costly iterations and utilizes a unique weighting scheme that is also developed in this research. The weighting scheme for assigning weights to point correspondences is independent of the complete state estimation system and can be easily used in other point registration applications. Making the proposed OD system more accessible by removing dependencies is essential for its future success. LiDAR is currently an expensive sensing solution; removing other costs such as labor for vehicle integration and other sensors makes the OD system more desirable.

All three objectives enumerated in Chapter 1 were completed. This thesis presents a proof of concept (PoC) of the importance map (IM) obstacle detection system. The IM system is designed for obstacle detection at highway speeds or in dynamic environments where the priority is to avoid the obstacle and not necessarily care about what is the obstacle. In these situations, it is simply not practical to predict object classifications. The processing for this system relies on a single frame of reference, the LiDAR frame, which makes IM an ideal choice for GPS-denied situations or for small vehicles, such as drones. IM with vehicle state estimation (IM+) only requires a powered LiDAR sensor and a processor. This new obstacle detection system is not a silver bullet, but at least provides a unique alternative for obstacle detection teams that need high OD performance in demanding environments with constrained processing resources.

5.1 Future Work

Despite the progress made in this work, there are many areas for improvement. With respect to the importance map, an effective, pixel-wise technique for eliminating road detections is needed. For this research, the code implementing the importance map and state estimation was not completely optimized. Future work could include optimizing the code and developing a GPU implementation. The obstacle mask created by the importance map contains noisy points. Techniques for removing noise and segmenting the obstacle mask

would be useful to improve the output of IM. With respect to state estimation, improving the accuracy of the current estimator would be valuable. Some ideas for improving this accuracy include updating the EKF dynamics model, adaptively finding the measurement variance from the number of available points, and handling open areas that do not have a lot of 3D features.

REFERENCES

[1] J. D. S. Wassim G. Najm and M. Yanagisawa, "Pre-crash scenario typology for crash avoidance research," Tech. Rep., 2007. [Online]. Available: https://www.nhtsa.gov/sites/nhtsa.gov/files/pre-crash_scenario_typology-final_pdf_version_5-2-07.pdf

[2] "2020 fatality data show increased traffic fatalities during pandemic," 2021. [Online]. Available: https://www.nhtsa.gov/press-releases/2020-fatality-data-show-increased-traffic-fatalities-during-pandemic

[3] F. Zhao, Q. Kong, Y. Zeng, and B. Xu, "A Brain-Inspired Visual Fear Responses Model for UAV Emergent Obstacle Dodging," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 12, no. 1, pp. 124–132, Mar. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8823938/

[4] X. Yu and M. Marinov, "A Study on Recent Developments and Issues with Obstacle Detection Systems for Automated Vehicles," *Sustainability*, vol. 12, no. 8, p. 3281, Apr. 2020. [Online]. Available: https://www.mdpi.com/2071-1050/12/8/3281

[5] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, p. 55, 1948.

[6] S. Singh and P. Keller, "Obstacle detection for high speed autonomous navigation," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation.* Sacramento, CA, USA: IEEE Comput. Soc. Press, 1991, pp. 2798–2805. [Online]. Available: http://ieeexplore.ieee.org/document/132057/

[7] Y. Li and J. Ibanez-Guzman, "Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50–61, 2020.

[8] S. Royo and M. Ballesta-Garcia, "An Overview of Lidar Imaging Systems for Autonomous Vehicles," *Applied Sciences*, vol. 9, no. 19, p. 4093, Sep. 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/19/4093

[9] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128$\times$128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008. [Online]. Available: http://ieeexplore.ieee.org/document/4444573/

[10] G. Chen, H. Cao, J. Conradt, H. Tang, F. Rohrbein, and A. Knoll, "Event-Based Neuromorphic Vision for Autonomous Driving: A Paradigm Shift for Bio-Inspired Visual Sensing and Perception," *IEEE Signal Process. Mag.*, vol. 37, no. 4, pp. 34–49, Jul. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9129849/

[11] S.-C. Liu and T. Delbruck, "Neuromorphic sensory systems," *Current Opinion in Neurobiology*, vol. 20, no. 3, pp. 288–295, Jun. 2010. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0959438810000450

[12] G. Gallego, T. Delbruck, G. M. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based Vision: A Survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–1, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9138762/

[13] R. R. Vyas, "Neuromorphic Retina Design to encode LIDAR based Scene Dynamics," Master's thesis, Delft University of Technology, 2019.

[14] M. Tsiourva and C. Papachristos, "LiDAR Imaging-based Attentive Perception," in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. Athens, Greece: IEEE, Sep. 2020, pp. 622–626. [Online]. Available: https://ieeexplore.ieee.org/document/9213910/

[15] A. Singh, A. Kamireddypalli, V. Gandhi, and K. M. Krishna, "Lidar guided small obstacle segmentation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*. IEEE, 2020, pp. 8513–8520. [Online]. Available: https://doi.org/10.1109/IROS45743.2020.9341465

[16] G. Gallego and D. Scaramuzza, "Accurate Angular Velocity Estimation With an Event Camera," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 632–639, Apr. 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7805257/

[17] G. Gallego, H. Rebecq, and D. Scaramuzza, "A Unifying Contrast Maximization Framework for Event Cameras, with Applications to Motion, Depth, and Optical Flow Estimation," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3867–3876, Jun. 2018, arXiv: 1804.01306. [Online]. Available: http://arxiv.org/abs/1804.01306

[18] B. J. Schachter, *Automatic Target Recognition, Fourth Edition*. SPIE, Mar. 2020. [Online]. Available: https://doi.org/10.1117/3.2542436

[19] S. Watanabe, *Knowing and Guessing: A Quantitative Study of Inference and Information*. New York: John Wiley Sons, 1969.

[20] C. B. Cornwall and S. E. Budge, "Biology-inspired lidar-centric obstacle detection," in *Autonomous Systems: Sensors, Processing and Security for Ground, Air, Sea and Space Vehicles and Infrastructure 2022*, vol. 12115. SPIE, 2022.

[21] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete mathematics: a foundation for computer science*, 2nd ed. Reading, MA: Addison-Wesley, 1994.

[22] C. B. Cornwall and S. E. Budge, "A neuromorphic approach to lidar point cloud processing," in *Autonomous Systems: Sensors, Processing, and Security for Vehicles and Infrastructure 2021*, vol. 11748. International Society for Optics and Photonics, 2021, p. 1174805.

[23] F. Chance, "Fast, Efficient Neural Networks Copy Dragonfly Brains," *IEEE Spectrum*, Jul. 2021. [Online]. Available: https://spectrum.ieee.org/fast-efficient-neural-networks-copy-dragonfly-brains

[24] N. Franceschini, J. M. Pichon, and C. Blanes, "From insect vision to robot vision," *Phil. Trans. R. Soc. Lond. B*, vol. 337, no. 1281, pp. 283–294, Sep. 1992. [Online]. Available: https://royalsocietypublishing.org/doi/10.1098/rstb.1992.0106

[25] S. Thrun, W. Burgard, and F. Dieter, *Probabilistic robotics*, ser. Intelligent robotics and autonomous agents. Cambridge, Massachusetts: The MIT Press, 2005.

[26] T. K. Moon and W. C. Stirling, *Mathematical methods and algorithms for signal processing*, 2000, no. 621.39: 51 MON.

[27] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[28] Y. Li and J. Ibanez-Guzman, "Lidar for autonomous driving: The principles, challeneges, and trends for automotive lidar and perception systems," *IEEE Signal Processing Magazine*, vol. 37, 2020.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[31] M. Thoma, "Analysis and optimization of convolutional neural network architectures," *arXiv preprint arXiv:1707.09725*, 2017.

[32] A. Banan, A. Nasiri, and A. Taheri-Garavand, "Deep learning-based appearance features extraction for automated carp species identification," *Aquacultural Engineering*, vol. 89, p. 102053, May 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0144860919302195

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[34] P. Wilmott, *Machine Learning: An Applied Mathematics Introduction*. Panda Ohana Publishing, 2019. [Online]. Available: https://books.google.com/books?id=f_WaxQEACAAJ

[35] A. O. Ly and M. Akhloufi, "Learning to Drive by Imitation: An Overview of Deep Behavior Cloning Methods," *IEEE Trans. Intell. Veh.*, vol. 6, no. 2, pp. 195–209, Jun. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9117169/

[36] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *arXiv preprint arXiv:1704.07911*, 2017.

[37] "Conv2d." [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

[38] R. Eisele, "Ackerman Steering • Computer Science and Machine Learning." [Online]. Available: https://www.xarg.org/book/kinematics/ackerman-steering/

[39] "Ros 2 documentation." [Online]. Available: https://docs.ros.org/en/dashing/index.html

[40] "Pytorch." [Online]. Available: https://pytorch.org

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[42] "Torch.nn.init." [Online]. Available: https://pytorch.org/docs/stable/nn.init.html

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[44] Marchel, C. Specht, and M. Specht, "Testing the Accuracy of the Modified ICP Algorithm with Multimodal Weighting Factors," *Energies*, vol. 13, no. 22, p. 5939, Nov. 2020. [Online]. Available: https://www.mdpi.com/1996-1073/13/22/5939

[45] P. S. Maybeck, *Stochastic Models: Estimation and Control: Volume 1*. Academic Press, 1979.

[46] P. Maybeck, *Stochastic Models, Estimation and Control*, ser. Mathematics in science and engineering. Academic Press, 1979, no. v. 2. [Online]. Available: https://books.google.com/books?id=6wdRAAAAMAAJ

[47] J. R. Carpenter and C. N. D'Souza, "Navigation filter best practices," Tech. Rep., 2018.

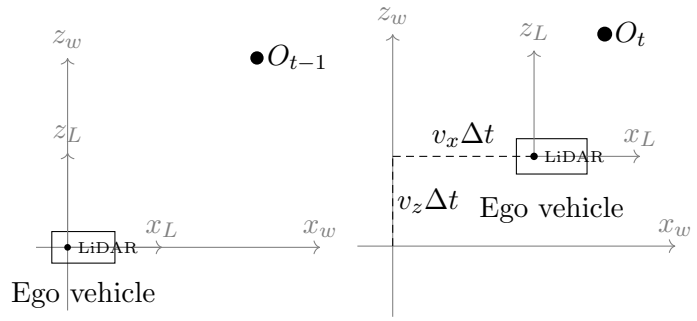
APPENDICES

APPENDIX A

Importance Map Creation

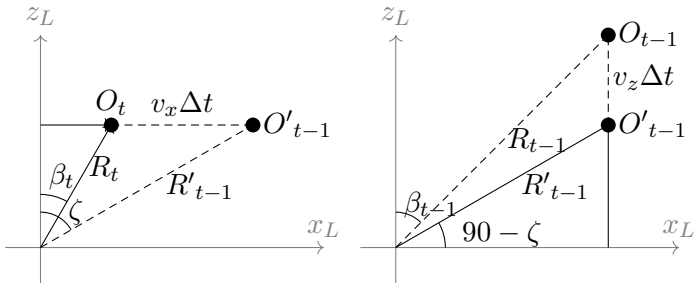
A.1 Static Object Relationships in the XZ Plane

Figures A.1 and A.2 explicitly show the relationship between a static object at time t and $t-1$ in the x-z plane. These are used to derive (2.13) and (2.14).



(a) (b)

Fig. A.1: Diagrams showing the position of the LiDAR and the static object in the x-z plane of the world frame when the vehicle is moving. (a) Vehicle at time $t-1$. (b) Vehicle at time t.



(a) (b)

Fig. A.2: Diagrams depicting the geometric relationship between a static object at t and the same static object at $t-1$ in the LiDAR frame. The ranges R shown are the ranges in the x-z plane. (a) Static object movement in the LiDAR frame in the x direction, as shown in A.1(b). (b) Static object movement in the LiDAR frame in the z direction.

APPENDIX B

Vehicle State Estimation

B.1 Weighted Point Registration

Let the error be formulated as

$$\mathbf{e}(k) = \sum_i \alpha_i ||\mathbf{p}_i(k) - R(k)\mathbf{p}_i(k-1) - \mathbf{t}(k)||_2^2, \tag{B.1}$$

where α_i is a real-valued number in the range $[0, 1]$. To minimize (B.1), set $\frac{\partial \mathbf{e}(k)}{\mathbf{t}(k)} = \mathbf{0}$, since $\mathbf{e}(k)$ is quadratic in $\mathbf{t}(k)$. This leads to

$$\mathbf{0} = \sum_i 2\alpha_i \Big[(\mathbf{p}_i(k) - R(k)\mathbf{p}_i(k-1) - \mathbf{t}(k) \Big]. \tag{B.2}$$

Solving (B.2) for $\mathbf{t}(k)$ gives

$$\hat{\mathbf{t}}(k) = \frac{1}{N_w} \sum_i \alpha_i \mathbf{p}_i(k) - R(k) \frac{1}{N_w} \sum_i \alpha_i \mathbf{p}_i(k-1), \tag{B.3}$$

where $N_w = \sum_i \alpha_i$. The scaled summations of points are the weighted averages of the two sets of points, which implies

$$\hat{\mathbf{t}}(k) = \boldsymbol{\mu}_w(k) - R(k)\boldsymbol{\mu}_w(k-1). \tag{B.4}$$

To minimize the error with respect to the translation $\mathbf{t}(k)$, substitute $\hat{\mathbf{t}}(k)$ for $\mathbf{t}(k)$ in the error term:

$$\mathbf{e}(k) = \sum_i \alpha_i ||\mathbf{p}_i(k) - R(k)\mathbf{p}_i(k-1) - \boldsymbol{\mu}_w(k) + R(k)\boldsymbol{\mu}_w(k-1)||_2^2. \tag{B.5}$$

Rearranging (B.5) gives

$$\mathbf{e}(k) = \sum_i \alpha_i ||\mathbf{p}_i(k) - \boldsymbol{\mu}_w(k) - R(k)\Big[\mathbf{p}_i(k-1) - \boldsymbol{\mu}_w(k-1)\Big]||_2^2. \tag{B.6}$$

Let $\bar{\mathbf{p}}_i(k) = \mathbf{p}_i(k) - \boldsymbol{\mu}_w(k)$. Therefore, the error can be expressed in terms of normalized points as

$$\mathbf{e}(k) = \sum_i \alpha_i ||\bar{\mathbf{p}}_i(k) - R(k)\bar{\mathbf{p}}_i(k-1)||_2^2. \tag{B.7}$$

The normed term in the argument of the summation in (B.7) can be expanded to

$$\bar{\mathbf{p}}_i(k)^T\bar{\mathbf{p}}_i(k) - 2\bar{\mathbf{p}}_i(k)^T R(k)\bar{\mathbf{p}}_i(k-1) + \bar{\mathbf{p}}_i(k-1)^T R(k)^T R(k)\bar{\mathbf{p}}_i(k-1). \tag{B.8}$$

The first term of (B.8) is constant with respect to $R(k)$, so it can be ignored. The last term can be simplified to $\bar{\mathbf{p}}_i(k-1)^T\bar{\mathbf{p}}_i(k-1)$ since $R(k)$ is unitary, making the last term unnecessary. By also removing the -2 constant in front of the second term, the point registration problem can now be explicitly expressed as

$$R(k) = \underset{R(k)}{\arg\max} \sum_i \alpha_i \bar{\mathbf{p}}_i(k)^T R(k)\bar{\mathbf{p}}_i(k-1). \tag{B.9}$$

By expressing (B.9) in terms of a trace, the expression to be maximized becomes

$$tr\Bigg(\begin{bmatrix}\alpha_1\bar{\mathbf{p}}_1(k)^T \\ \alpha_2\bar{\mathbf{p}}_2(k)^T \\ \vdots \\ \alpha_n\bar{\mathbf{p}}_n(k)^T\end{bmatrix} R(k) \begin{bmatrix}\bar{\mathbf{p}}_1(k-1) & \bar{\mathbf{p}}_2(k-1) & \dots & \bar{\mathbf{p}}_n(k-1)\end{bmatrix}\Bigg), \tag{B.10}$$

which can be rewritten as

$$tr(P_w(k)^T R(k) P(k-1)), \tag{B.11}$$

where $P(k) = \begin{bmatrix} \bar{\mathbf{p}}_1(k) & \bar{\mathbf{p}}_2(k) & \dots & \bar{\mathbf{p}}_n(k) \end{bmatrix}$ and $P_w(k) = \begin{bmatrix} \alpha_1\bar{\mathbf{p}}_1(k) & \alpha_2\bar{\mathbf{p}}_2(k) & \dots & \alpha_n\bar{\mathbf{p}}_n(k) \end{bmatrix}$. From the cyclical commutation property of the trace, (B.11) becomes

$$tr(P(k-1)P_w(k)^T R(k)). \tag{B.12}$$

Equation B.12 can then be expressed as

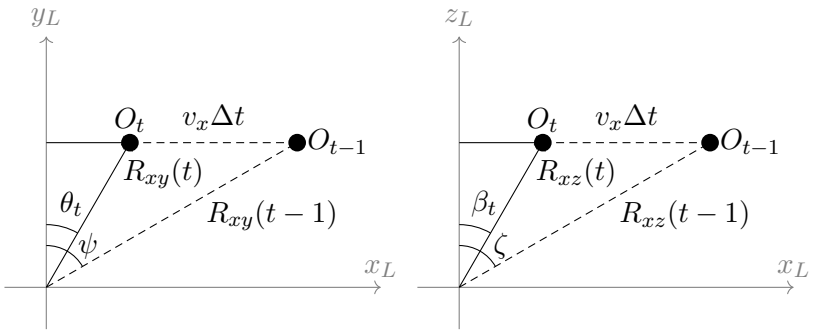
$$tr(U\Sigma V^T R(k)) = tr(\Sigma V^T R(k) U), \tag{B.13}$$

using the cyclical commutation property and where $SVD(P(k-1)P_w(k)^T) = U\Sigma V^T$. Since the singular values of $P(k-1)P_w(k)^T$ should be greater than zero since $P(k-1)P_w(k)^T$ should be full-rank and using the fact V^T, $R(k)$, and U are all unitary, $V^T R(k) U$ must be an identity matrix to maximize (B.13). Therefore,

$$\hat{R}(k) = VU^T. \tag{B.14}$$

B.2 Predict Previous Range of Static Object

Only the longitudinal velocity of the vehicle v_x will be considered since all other translational velocities are negligible. The angular velocity of the vehicle is also not considered because angular rotation does not affect the distance of points from the vehicle. Figure B.1 shows the relationship between the current range and the past range of a static object.



(a) (b)

Fig. B.1: Diagrams depicting the geometric relationship between a static object at t and the same static object at $t-1$ in the LiDAR frame, ignoring all translational velocities except longitudinal (x) velocity. (a) Static object movement in the LiDAR frame in the x-y plane. (b) Static object movement in the LiDAR frame in the x-z plane.

Using Figure B.1,

$$R_{xy}(t-1) = \frac{R_{xy}(t)cos(\theta_t)}{cos(\psi)} \tag{B.15}$$

and

$$R_z(t-1) = R_{xz}(t-1)cos(\zeta) = R_{xz}(t)cos(\beta_t). \tag{B.16}$$

From (B.15) and (B.16), the past range is

$$R_{t-1} = \sqrt{R_{xy}(t-1)^2 + R_z(t-1)^2}. \tag{B.17}$$

The value R_{t-1} is equivalent to R_{pred}, which is used in Section 4.1.2.