A CAMERA-LIDAR FUSION FRAMEWORK

A LIGHTWEIGHT CAMERA-LIDAR FUSION FRAMEWORK FOR TRAFFIC MONITORING APPLICATIONS

BY

ADRIAN SOCHANIWSKY, B.Eng.

A Thesis

submitted to the Computing and Software

and the School of Graduate Studies

of McMaster University

in partial fulfilment of the requirements

for the degree of

Master of Applied Science

Master of Applied Science (2024) McMaster University

(Computing and Software) Hamilton, Ontario, Canada

TITLE:	A Lightweight Camera-LiDAR Fusion Framework for Traffic Monitoring Applications
AUTHOR:	Adrian Sochaniwsky B.Eng. (Mechatronics Engineering), McMaster University, Hamilton, Canada
SUPERVISOR:	Dr. Martin von Mohrenschildt and Dr. Saeid Habibi
NUMBER OF PAGES:	xvii, 139

Lay Abstract

Accurate traffic monitoring systems are needed to improve the safety of road users. These systems allow the intersection to "see" vehicles and pedestrians, providing near instant information to assist future autonomous vehicles, and provide data to city planers and officials to enable reductions in traffic, emissions, and travel times. This thesis aims to design, build, and test a traffic monitoring system that uses a camera and 3D laser-scanner to find and track road users in an intersection. By combining a camera and 3D laser scanner, this system aims to perform better than either sensor alone. Furthermore, this thesis will collect test data to prove it is accurate and able to see vehicles and pedestrians during the day and night, and test if runs fast enough for "live" use.

Abstract

Intelligent Transportation Systems are advanced technologies used to reduce traffic and increase road safety for vulnerable road users. Real-time traffic monitoring is an important technology for collecting and reporting the information required to achieve these goals through the detection and tracking of road users inside an intersection. To be effective, these systems must be robust to all environmental conditions. This thesis explores the fusion of camera and Light Detection and Ranging (LiDAR) sensors to create an accurate and real-time traffic monitoring system. Sensor fusion leverages complimentary characteristics of the sensors to increase system performance in low-light and inclement weather conditions. To achieve this, three primary components are developed: a 3D LiDAR detection pipeline, a camera detection pipeline, and a decision-level sensor fusion module. The proposed pipeline is lightweight, running at 46 Hz on modest computer hardware, and accurate, scoring 3% higher than the camera-only pipeline based on the Higher Order Tracking Accuracy metric. The camera-LiDAR fusion system is built on the ROS 2 framework, which provides a well-defined and modular interface for developing and evaluated new detection and tracking algorithms. Overall, the fusion of camera and LiDAR sensors will enable future traffic monitoring systems to provide cities with real-time information critical for increasing safety and convenience for all road-users.

Acknowledgements

First, I would like to express gratitude to my advisors, Dr. Martin von Mohrenschildt and Dr. Saeid Habibi for providing the opportunity to pursue this Master's degree. I thank Dr. von Mohrenschildt for the thought-provoking conversations and his endless enthusiasm as an educator and researcher. I also thank Dr. Habibi for his experience, guidance, and resources at CMHT that have allowed me to pursue this project.

I am grateful for all my friends and colleagues at CMHT and McMaster for the advice and discussions over the last two years. Especially Dr. Yixin Huangfu, who was always available to answer my questions and provide suggestions.

Finally, I am thankful for my partner, Maya, for all her support and encouragement. And to my parents for their support and unwavering belief in my abilities.

Contents

List of Figures

List of Tables

Abbreviations

AABB	Axis-Aligned Bounding Box
AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Average Precision
AV	Autonomous Vehicle
BEV	Bird's Eye View
CMHT	Centre for Mechatronics and Hybrid Technologies
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
FoV	Field of View
FPS	Frames per Second

GPU	Graphics Processing Unit
HOG	Histogram of Oriented Gradients
HOTA	Higher Order Tracking Accuracy
IoU	Intersection over Union
IR	Infrared
ITS	Intelligent Transportation Systems
KF	Kalman Filter
LiDAR	Light Detection and Ranging
mAP	Mean Average Precision
MARC	McMaster Automotive Resource Centre
ML	Machine Learning
MOT	Multi-Object Tracking
MOTA	Multi-Object Tracking Accuracy
MOTP	Multi-Object Tracking Precision
NIR	Near Infrared
NMS	Non-Maximum Suppression
OBB	Oriented Bounding Box
PANet	Path Aggregation Network

PCA	Principal Component Analysis
PF	Particle Filter
R-CNN	Region based CNN
RADAR	Radio Detection and Ranging
ReLU	Rectified Linear Unit
RGB	Red, green, blue
RoI	Region of Interest
ROS	Robot Operating System
RPN	Region Proposal Network
SORT	Simple Online Real-time Tracking
SPP	Spatial Pyramid Pooling
SVM	Support Vector Machine
SVSF	Smooth Variable Structure Filter
ToF	Time-of-Flight
V2X	Vehicle-to-Everything
YOLO	You Only Look Once

Chapter 1

Introduction

Smart cities are urban areas that use advanced technology to improve public services such as transportation, healthcare, and safety [16]. One of the technologies to achieve these goals is Intelligent Transport Systems (ITS), which provide transportation and traffic services. By collecting real-time data, smart cities can improve traffic flow by detecting traffic congestion, reduce travel time, and create safer intersections by identifying accidents, near-misses, and illegal manoeuvres. Accurate, real-time detection and tracking are necessary to provide this valuable information to city planners and officials so they can monitor and validate the impact of their decisions.

One type of ITS is a roadside perception system that can accurately detect and track objects to monitor traffic and output road-user location and speed, and intersection occupancy. This information can be shared to accelerate and support technology such as the autonomous vehicle (AV). Stationary traffic monitoring devices can share information to help solve the over-the-horizon issue and make roads safer by providing information outside the perception of a singular vehicle [17]. The roadside perception systems will be a critical component of the vehicle-to-everything (V2X)

concept, which utilizes distributed information from multiple sources external to an autonomous vehicle to increase its safety and ability to function in complex driving scenarios. To effectively assist AVs, the roadside system must be robust to changing lighting and weather conditions, and process data in real-time. Figure 1.1 visualizes the V2X concept, information is shared between all users of an intersection to maximize perception and planning capabilities.

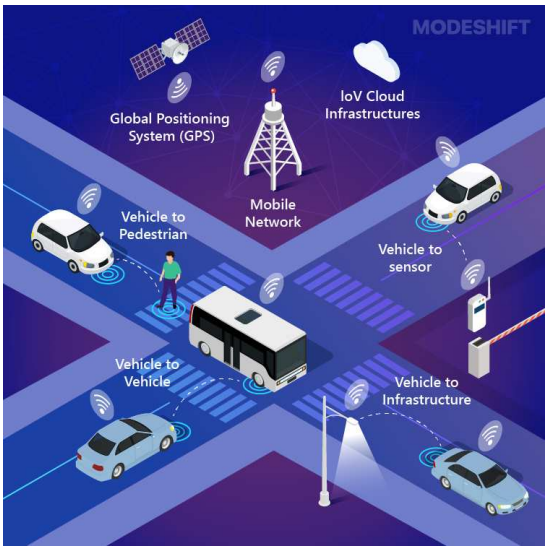


Figure 1.1: Visual representation of the V2X concept [1].

Smart cities and traffic monitoring systems will enhance safety, equity, and inclusion for vulnerable road users and the public. Aside from assisting AVs, these systems can be combined with smart traffic lights [18] to improve traffic flow based on live measurements of traffic density. By dynamically adjusting the timing and synchronization of street lights, queue lengths and congestion can be reduced. This will also improve air quality and commute times for all road users in the city. Long-term statistical trends of city traffic can be aggregated to assess the effectiveness of city policies related to road safety and congestion. Having automated data-collection will

provide data for governments and researchers to model and design better solutions for traffic related problems.

A smart city can have thousands of intersections equipped with traffic monitoring sensors, each producing massive data streams. Thus, the monitoring of intersections should be an online process, where sensor data is processed via edge devices and only incidents and aggregated intersection statistics are saved for offline processing and querying. Since edge devices are located near the sensors that provide the data streams, the financial cost of the computational hardware should also be minimized due to the number of devices required. This poses an additional constraint on the underlying algorithms, which need to process the data online while respecting the computational limits of the hardware. Another benefit to online data processing is that object detection results preserve road-user privacy, since aggregated results rather than raw data are transmitted to a database.

Many commercial solutions for traffic monitoring use one or more cameras per intersection. Vision-based systems offer a mature and low-cost solution with high accuracy in good lighting conditions. However, poor lighting conditions and inclement weather reduce their performance. An event as common as nighttime snowfall can drastically reduce the accuracy and range of these systems. This has led to interest in Light Detection and Ranging (LiDAR) or combining multiple sensors through sensor fusion.

Sensor fusion merges information from multiple sensors to generate a more accurate output than either sensor alone. This can include multiple sensors of the same type or two or more different types of sensors. The goal is to combine complimentary characteristics from each sensor to provide better accuracy and increased robustness

in difficult environmental conditions and complex scenarios. Common object detection and tracking sensors include cameras, LiDAR sensors, and Radio Detection And Ranging (RADAR).

This thesis will explore a camera-LiDAR sensor fusion system for traffic monitoring, with an emphasis on computationally efficient and modular software.

1.1 Motivation and Objectives

LiDAR sensors are increasing in popularity for many applications requiring high-accuracy three-dimensional measurements. Once restricted to research and development projects, LiDAR is successfully used in robotics, mapping, surveillance, and AVs, further advancing its adoption via increased production volume and reduced costs. The properties of the LiDAR sensor make it a strong candidate for traffic monitoring applications.

The primary objective of this thesis is to design, implement, and test an online multi-object tracking (MOT) system based on camera images and LiDAR point cloud data for static traffic monitoring applications. The system processes live LiDAR data directly from the sensors and outputs position, velocity, size, and a tracking identifier for each object in the region of interest (RoI). The system runs at real-time on modest computer hardware. Additional objectives include:

- Design and assemble a test fixture for data collection, including a LiDAR and camera sensor.
- Capture a synchronized dataset of LiDAR point clouds and images in multiple lighting scenarios.

- Design and implement a modular LiDAR data processing pipeline.
- Implement a baseline algorithm for fusing camera and LiDAR object detections
- Create a set of supporting tools for deploying the system in a real environment.

1.2 Contributions

The main contributions of this thesis include:

1. Design of an end-to-end Camera-LiDAR fusion MOT pipeline for traffic monitoring applications. Algorithms are selected to achieve real-time performance on modest computer hardware.
2. Implementation of a modular framework with a clearly defined interface for ease of experimentation. Algorithms can be swapped and run on live or pre-recorded data.
3. Development of a test fixture for collection of a synchronized LiDAR point cloud and image dataset. The images are labelled with object detections and track IDs for pipeline evaluation.

1.3 Overview

This thesis is structured as follows. Chapter 2 provides an introduction and review of sensor technology for object detection and a comparison between camera, RADAR, and LiDAR sensors. Then, sensor selection and calibration are discussed. Finally,

datasets are discussed and the CMHT Traffic dataset is introduced. Chapter 3 reviews neural networks for image classification and object detection. Then discusses the implementation and evaluation of a camera-based detection pipeline. Chapter 4 provides a literature review of 3D object detection in LiDAR point clouds and then details the theory and implementation of the object detection module. Chapter 5 reviews LiDAR object classification and then discusses the algorithm design. Chapter 6 includes a literature review of multi-object tracking and then explains the data association and tracking method. Then camera and LiDAR tracking pipelines are evaluated. Chapter 7 provides a literature review of sensor fusion approaches and then details the camera-LiDAR fusion method. Chapter 8 details the implementation of the system, including the frameworks and tools used. Chapter 9 provides an overview of accuracy and runtime performance evaluations. Chapter 10 provides concluding remarks and possible future work.

Chapter 2

Sensors and Calibration

2.1 Introduction

This chapter will explore the role of sensors in MOT systems. Sensors convert information from their environment into signals that can be consumed and processed by a computer algorithm. The choice of sensor(s) is critical to system performance since each type of sensor varies in accuracy, cost, and robustness. Camera, RADAR, and LiDAR are common sensors that are currently used for traffic monitoring and other systems that require accurate tracking of objects. This chapter will begin by examining each sensor, including strengths, weaknesses, and example output. Then comparisons will be drawn between each type of sensor. The chapter will conclude with a discussion on datasets and data collection.

There are many ways to categorize each sensing modality, such as the dependence on external energy for capturing measurements. Passive sensors function on the energy that already exists in the environment, and active sensors emit energy to capture measurements.

2.2 Cameras

A camera is a passive sensor that can be classified by the wavelength that it measures, and can be used in multiple configurations, such as monocular or stereo. Modern digital cameras use a lens to focus light onto an image sensor that converts the light into an electrical signal. The signals vary based on intensity and colour and are processed into a digital image. A digital image is a two-dimensional array of pixels (picture elements). The pixels are usual 8-bit unsigned integers that represent the discretized amplitude at that point. Images contain rich information including colour, texture, shape, and context of objects. Images are also dense because they have a high resolution. Challenges in image processing include viewpoint variation, occlusion, scale, deformation, and motion blur. While these problems can be trivial for humans, computer algorithms can be significantly impacted if not designed properly. Figure 2.1 demonstrates the effect of scale, where two objects with the same physical size are scaled based on their distance from the sensor, posing an additional challenge for detecting objects as the same object is represented with a different number of pixels depending on its distance.

Figure 2.1: Example of different object scales in an image when objects are approximately the same size in 3D space.

Red, green, and blue (RGB) cameras use digital optical components and colour filters to convert the visible spectrum of light entering the sensor into images with three channels for red, blue, and green colours. These tristimulus colours are a standardized method of representing 2^{24} colours via three 8-bit channels. These sensors are inexpensive, produce high-resolution images, and are capable of high frame rates. These characteristics make RGB cameras a popular choice for many applications. However, they perform poorly in low light conditions and inclement weather such as fog and snow, since these conditions limit the amount of energy that the camera can passively measure. Figure 2.2 demonstrates the blooming effect when a bright light source is present in a dark scene, this can cause an image-based object detector to fail. Additionally, extremely bright scenes or large changes in lighting can produce artifacts including shows, highlights, or blurring. Furthermore, images are inherently two-dimensional and provide no depth information.

Figure 2.2: Example of camera sensitivity to changes in lighting, shown by vehicle headlamps.

2.2.1 Infrared and Thermal Cameras

IR cameras function similarly to RGB cameras, operating in a different part of the light spectrum, and detecting near-infrared (NIR) light reflected from objects. Figure 2.3 illustrates the differences in wavelength for RGB and IR cameras and provides an example image from each. IR cameras function better in low-light conditions and are often combined with IR light-emitting diodes (LEDs) for nighttime applications. Since the infrared wavelength is larger than visible light, the image sensor must use larger detectors, resulting in fewer pixels per image. This leads to lower resolutions and a lower sensitivity compared to RGB cameras. Thermal cameras measure emitted in the medium and long IR range and suffer from additional issues, such as thermal crossover and blooming. Thermal crossover occurs when unique objects have a negligible temperature delta and differentiation between these objects becomes difficult. Thermal blooming occurs when an object or the environment emits heat, causing blurring and distortion of the image.

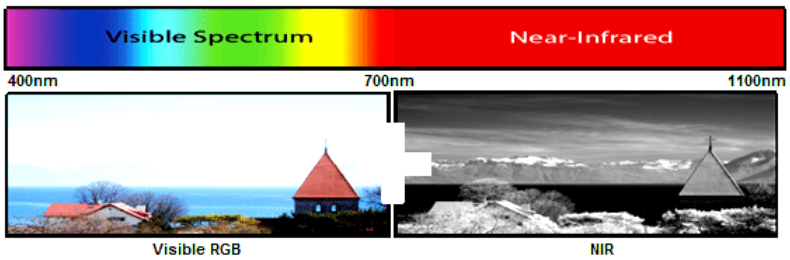


Figure 2.3: Wavelengths of the RGB and NIR cameras [2].

2.2.2 Depth Cameras

A monocular camera system uses a single camera sensor for capturing images. The image sensor only records a two-dimensional projection that lacks depth information. Since object tracking is inherently a three-dimensional problem, it can be more difficult to accurately localize three-dimensional objects in the image plane.

In addition to RGB channels, depth cameras provide an additional channel of depth information. Common methods for acquiring a depth image include structured light, time-of-flight (ToF), and stereo cameras. Structured light and ToF cameras require a precise external source of light to determine depth information, these approaches are affected by lighting conditions and surface material. Stereo cameras function much like the human eyes, by capturing the scene with two precisely offset perspectives. Since the exact positioning of the lenses is known, the stereo photography technique can be used to create a depth image that includes a third dimension for each pixel. All depth cameras have a limited working range, which limits their usefulness for outdoor detection applications.

2.3 LiDAR

LiDAR is an active sensing technology that uses laser beams to accurately measure distance. The sensor uses the reflection of the laser beam from distant objects to measure the range and other properties such as reflectance and intensity [19].

LiDAR acquires 3-dimensional laser scans of their environment by sweeping an array of laser beams. The number of beams determines the vertical resolution, thus affecting the scan density and increasing the amount of data processing required.

Common values include 32, 64 and 128 beams.

A point cloud is a set of points with discrete Cartesian x, y, z values. For LiDAR sensors, each laser beam measures a single point that contains the (x, y, z) position. One sweep of the laser is called a frame, and collectively, the points within the frame form a point cloud. The size of the point cloud size is dictated by the vertical and horizontal resolution of the sensor. Some LiDAR provide additional information per point, such as, intensity and reflectance. Figure 2.4 shows an example point-cloud where the points are coloured using the intensity data.

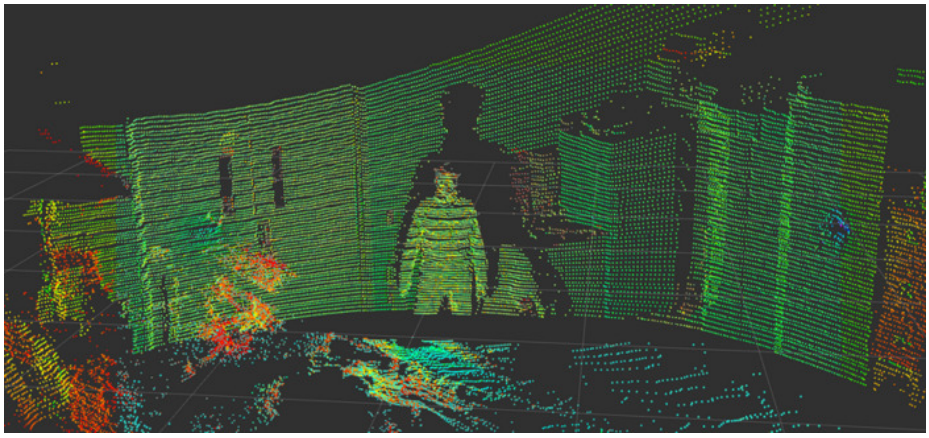


Figure 2.4: Example of a point cloud generated from a LiDAR sensor.

LiDAR data can be in the form of an organized or unorganized point cloud. Organized point clouds use the x, y values to create a multidimensional array similar to images, which allows algorithms to take advantage of the spatial relation between points in the array. Unorganized point clouds contain all the points in a 1-dimensional vector.

Since LiDAR produces an accurate 3D representation of their environment, objects are represented by an approximation of their true size regardless of the distance from the sensor. However, since the angle of the laser beams is fixed, the density of the

point cloud decreases proportionally to the distance of the objects being measured. Figure 2.5 demonstrates this, when the same vehicle is measured further away from the sensor, the number of measurements decreases from 7 to 3.

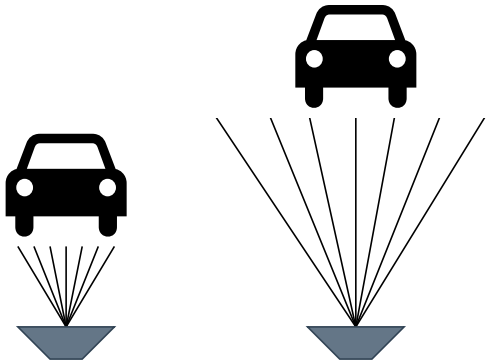
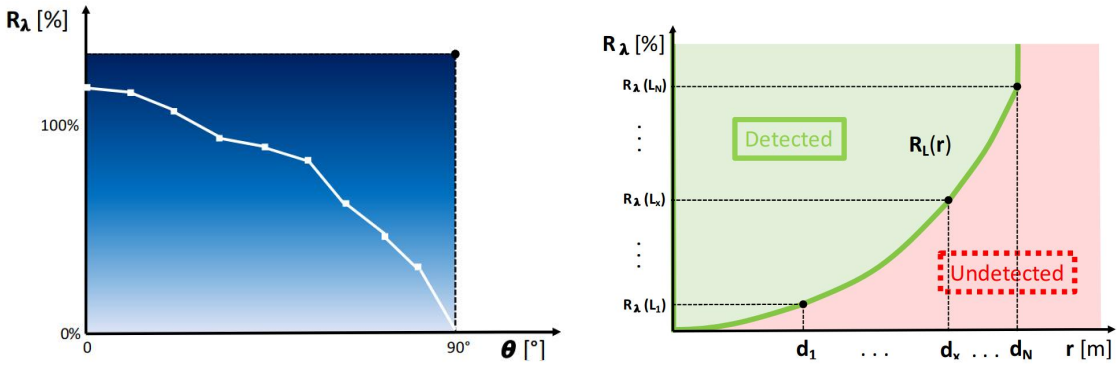


Figure 2.5: Visualization of beam divergence for a LiDAR sensor. The same size object will be represented by a number of points inversely proportional to the distance from the sensor.

The type of material effects an object's reflectivity and LiDAR detectability. Additionally, the angle of the object's surface to the sensor has a strong influence on detection. Figure 2.6a exemplifies the reflectance of a material, R_λ as a function of the angle of incidence θ. The curve shows that objects more perpendicular with the sensor will have a higher reflectivity. Figure 2.6b demonstrates that for a range of distances, the reflectance of an object is critical to its detection.



(a) Exemplary reflectance R_λ curve depending on the incidence angle θ [20].

(b) Exemplary reflectance limits function $R_{L(r)}$, depending on range r [20].

Figure 2.6: The effect of reflectivity on LiDAR performance.

2.3.1 Mechanical LiDAR

Mechanical LiDAR sensors operate by rotating internal components to acquire a 360-degree scan. The field-of-view (FoV) of a mechanical LiDAR can be seen in Figure 2.7, where 45 degrees in the vertical plane is swept 360 degrees horizontally. Since it has moving parts, mechanical LiDAR is heavier, uses more power, and tends to have a larger package size. Figure 2.7 is an Ouster OS1-64. The sensor has 64 laser beams, and takes 2048 steps for each rotation, producing a laser scan with 2048×64 points. LiDAR point cloud density is relatively low compared to an HD camera with 1920×1080 resolution, but has a higher density than a RADAR.

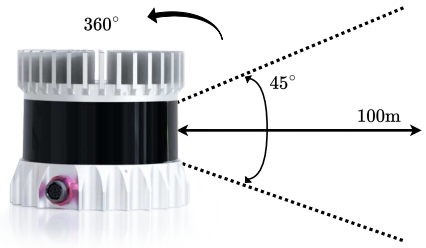


Figure 2.7: Diagram of the FoV of a mechanical LiDAR.

2.3.2 Solid-State LiDAR

Solid-state LiDAR uses micro-electromechanical systems (MEMS) technology to acquire 3D scans without conventional moving parts. The benefit of using MEMS includes reducing weight, size, and power consumption. The FoV of solid-state LiDAR sensors vary depending on the task they are designed for. Some models offer a configurable resolution to balance density and computational effort. Solid-state LiDAR are capable of significantly denser point cloud output. Figure 2.8 shows the Blickfeld Cube 1 solid-state LiDAR sensor, which offers a maximum of 400 vertical beams, a configurable resolution, and a FoV of 70×40 degrees. Due to its lower energy consumption, the Blickfeld Cube 1 uses Power-Over-Ethernet to transmit data and provide power using a single cable.

Figure 2.8: Blickfeld Cube 1 outdoor solid-state LiDAR.

2.3.3 Laser Sources and Measurement

There are two popular wavelengths used in LiDAR laser sources, 905 and 1550 nanometers. Table 2.1 details the differences between them. Furthermore, there are two methods of measuring distance, ToF and Frequency modulated continuous wave (FMCW).

Table 2.1: Comparison of LiDAR laser wavelengths.

Property	**905 nm**	**1550 nm**
Cost	Lower	Higher
Range	Lower	Higher
Power Consumption	Lower	Higher
Accuracy	Lower	Higher
Weather Penetration	Better	Worse
Maturity	Older	Younger
Maximum Permissible Exposure	Lower	Higher
Best Use Case	Shorter range	Long-Range, fair-weather

Time of Flight

Pulsed ToF LiDAR use discrete pulses of a single-frequency laser to generate a distance measurement. A pulse is emitted from the sensor, and the time for it to bounce back to the receiver is recorded. The distance is calculated as $d = \frac{ct}{2}$, where c is the speed of light. Figure 2.9 shows how the emitted and returned beam travels $2d$. Given the distance and vertical and horizontal angles of the laser beam, the 3D Cartesian coordinate can be determined. ToF technology is used in both mechanical and solid-state sensors.

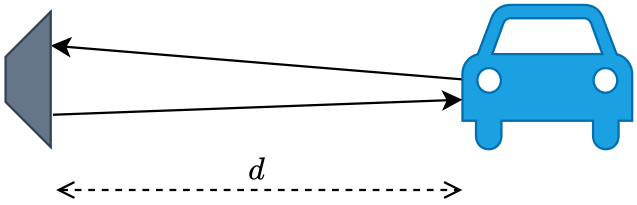


Figure 2.9: A diagram of the ToF method.

Frequency Modulated Continuous Wave

Frequency-modulated continuous wave (FMCW) LiDAR sensors uses a continuous coherent laser source which varies its frequency [21]. The received waves are compared to the original signal to determine the distance and velocity for each point. Thus, FMCW LiDAR are four-dimensional sensors that produce a (x, y, z, v) measurement for each point and may include additional data such as intensity or reflectivity. The spatial density of these sensors can be significantly greater than standard ToF-based LiDAR. FMCW is the more expensive technology since it is less mature and requires more complex hardware.

2.4 RADAR

RADAR is an active sensing technology that uses electromagnetic frequencies, typically between 76 and 81 GHz, to measure speed and distance. For applications that perform dynamic object tracking and detection, such as autonomous vehicles or traffic monitoring, mm-wave RADAR is a popular choice [22]. RADAR produces a (x, y, z, v) per point, where v is the radial velocity calculated from the Doppler effect. RADAR point clouds have a lower density relative to LiDAR due to a lower angular resolution. RADAR is robust to lighting because mm-waves are invariant to changes

in ambient lighting conditions. It is also robust to weather conditions since mm-wave RADAR is not attenuated by moisture or particulates in the air such as smoke or dust [23]. A drawback to RADAR is the clutter and noise the sensor produces, which can lead to false defections. Furthermore, most commercially available RADAR produce outputs for moving objects only, which complicates the detection of static objects in a traffic scene, such as a stopped vehicle.

2.5 Sensor Comparison

Figure 2.10 shows the characterization of three common sensors used for object detection and tracking. From the shape of the graph, it can be seen that combining camera and RADAR or camera and LiDAR would be beneficial. The density and richness in digital images are complimentary to the sparser RADAR and LiDAR point clouds, which provide 3D measurements and depth.

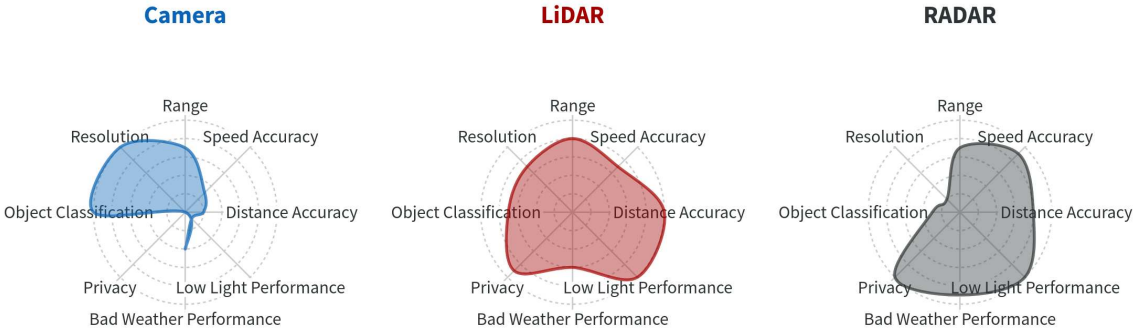


Figure 2.10: A comparison of Camera, RADAR, and LiDAR.

2.6 Sensor Selection

This section will describe the sensors that have been selected for this thesis.

2.6.1 LiDAR

An Ouster OS1-64 LiDAR sensor was used due to its technical specifications and availability in the Centre for Mechatronics and Hybrid Technologies (CMHT) laboratory. It has 64 beams with a maximum resolution of 2048 × 64 points, and a maximum range of 120 meters. Table 2.2 provides the full technical specifications of the sensor. An important consideration is the sensor range, as the maximum range can be affected by the target reflectivity and angle of the target surface relative to the sensor. Thus, the Ouster OS1 range will be sufficient for capturing city intersections.

Table 2.2: Specifications for the Ouster OS1-64 Gen. 1 high resolution imaging LiDAR [14].

Specification	**Value**
Range	120 m @ 80% reflective Lambertian target 40 m @ 10% reflective Lambertian target
Range Resolution	1.2 cm
Vertical Resolution	64
Horizontal Resolution	512, 1024, 2048 (configurable)
Field of View	Vertical: +16.6 to -16.6 (32.2 deg) Horizontal: 360 deg
Rotation Rate	10 to 20 Hz (configurable)
# of Returns	1 (strongest)
Laser Wavelength	850 nm
Points per second	1,310,720
Data Per Point	Range, intensity, ambient, reflectivity, angle, timestamp
Data Connection	UDP over gigabit Ethernet

(a) The *Ouster OS1-64* used for data collection with a synchronized camera mounted on a tripod.

(b) The sensor in the testing location.

Figure 2.11: The experimental setup.

2.6.2 Camera

A Logitech Brio HD Webcam is used to collect image data of the city intersections. Table 2.3 outlines the resolution, FoV, maximum frame rate, and other specifications. This camera was selected for its size, simple USB interface, and universal mounting option. The proposed fusion system uses images with a resolution of 1920×1280 collected at 10 Hz.

Table 2.3: Specifications for the Logitech Brio HD Webcam.

Specification	Value
Resolution	4090 × 2160 (30 FPS) 1920 × 1080 (30, 60 FPS) 1280 × 720 (30, 60, 90 FPS)
Field of View	Diagonal: 90 deg
Focus Type	Autofocus
Mounting Type	1/4”-20 thread
Weight	64 grams

2.6.3 Sensor Platform

Figure 2.11a shows the experimental setup that includes an Ouster OS1-64 LiDAR sensor and Logitech Brio camera to collect synchronized images and point clouds. Both camera and LiDAR capture data at 10 Hz, although they are not explicitly synchronized via hardware or software trigger. The timestamps between the image and point cloud pair are approximately 10 ms. Figure 2.11b shows the data collection location on top of the McMaster Automotive Resource Centre (MARC) building, approximately 12 m above the road surface, the sensors are rigidly mounted to a tripod for ease of repositioning and data collection.

2.7 Calibration

This section provides an introduction to sensor calibration and its applications. Intrinsic calibration relates to the internal parameters of the sensors. And extrinsic

calibration is the relation between sensor coordinate systems.

2.7.1 Intrinsic — Camera

Intrinsic camera calibration is required to localize the camera in 3D space and to project 3D points into the 2D image plane. The intrinsic calibration is not dependent on the scene and thus can be calculated once for a given camera and fixed resolution.

The pinhole camera model provides a first-order approximation for the relation between a point in 3D space and the 2D image plane. Equation 2.7.1 defines the relation between a 2D point in the image, p, and a 3D point in the camera frame, P_c, via the camera projection matrix, K. It assumes the camera that is used to take the image follows the ideal pinhole model, and it does not take into account distortion or blurring caused by the sensor. For most computer vision applications, this assumption does not have a large impact on the accuracy of the final result. Figure 2.12 depicts the pinhole camera model, where f is the distance along the optical axis from the origin to the image plane.

$$p = KP_c \tag{2.7.1}$$

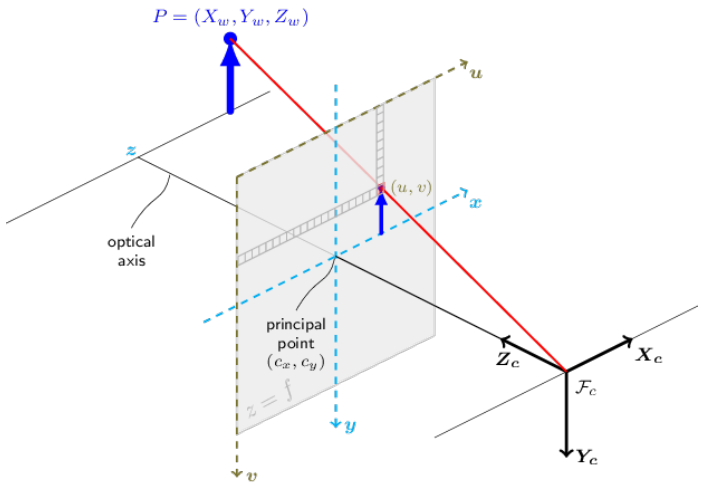


Figure 2.12: The pinhole camera model [3].

Since the optical image sensor is not perfectly aligned with the origin of the image, the camera projection matrix, K, takes the form defined in Equation 2.7.2 with $\alpha = fm_x$ and $\beta = fm_y$. The variables m_x and m_y are the ratio of pixels per unit of distance. And c_x, c_y is the translation to align the optical centre with the origin of the reference frame.

$$K = \begin{pmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.7.2}$$

A common calibration procedure involves taking multiple images with a checkerboard pattern of known dimensions in multiple orientations. The edges and corners are detected, and with the assumption that the board has perfect lines, the distortion within the image can be calculated. OpenCV [3] provides a convenient method of camera calibration. Intrinsic camera calibration only needs to be performed once per camera per the selected resolution.

2.7.2 Intrinsic - LiDAR

LiDAR sensors are calibrated at the factory, and most algorithms using LiDAR data do not need access to these parameters. However, it is common for firmware to support functions for querying these values [14].

2.7.3 Extrinsic

In roadside traffic monitoring systems, a sensor can be mounted at arbitrary positions overlooking an intersection. Furthermore, systems that fuse multiple sensor data require the extrinsic transformation between sensor origins, since the sensors can be arbitrarily mounted relative to one another. Extrinsic calibration refers to obtaining the rotation and translation components of a homogeneous transformation matrix between the reference frames or coordinate systems.

For cameras, the pixel coordinate system has the origin in the top left corner of the image, using the Z-axis out of the image plane, the X-axis for horizontal measurement and the Y-axis for vertical measurement. Figure 2.12 illustrates the pixel coordinate system as shown by u, v. The camera coordinate system is denoted by $\boldsymbol{X_c}, \boldsymbol{Y_c}, \boldsymbol{Z_c}$, and a 3D point in the world coordinate frame as X_w, Y_w, Z_w.

The LiDAR coordinate system follows the right-hand rule and uses the Z-axis for height, the X-axis for the front of the sensor and the Y-axis for the left face of the sensor. Figure 2.13 shows both sensors and the relative position between the coordinate frames. The transformation, ${}^{L}T_{C}$, defines the transformation of the camera frame relative to the LiDAR frame. Equation 2.7.5 shows how ${}^{L}T_{C}$ is constructed, measurements from the physical system provide the values for each parameter.

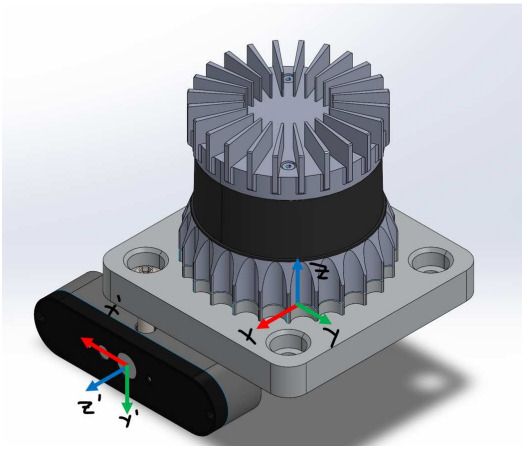


Figure 2.13: The coordinate frames for the camera (left) and LiDAR (right).

Two rotations and a translation are required to align the sensor origins

$$
\begin{aligned}
{}^{L}T_{C} =& \text{Rot}(X,\theta_1) \times \text{Rot}(Z,\theta_2) \times \text{Trans}(d_x, d_y, d_z) && (2.7.3)\\
=& \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\theta_1 & -S\theta_1 & 0 \\ 0 & S\theta_1 & C\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & 0 \\ S\theta_2 & C\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} && (2.7.4)\\
=& \begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & C\theta_2 d_x - S\theta_2 d_y \\ C\theta_1 S\theta_2 & C\theta_1 C\theta_2 & -S\theta_1 & C\theta_1 S\theta_2 d_x + C\theta_1 C\theta_2 d_y - S\theta_1 d_z \\ S\theta_1 S\theta_2 & S\theta_1 C\theta_2 & C\theta_1 & S\theta_1 S\theta_2 d_x + S\theta_1 C\theta_2 d_y + C\theta_1 d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} && (2.7.5)
\end{aligned}
$$

where θ_1, θ_2 are rotation angles and d_x, d_y, d_z are the translation components.

2.8 Datasets

Although traffic monitoring systems are designed for online data processing, recorded datasets are useful for development and comparing algorithm performance.

2.8.1 Public Datasets

Public datasets with stationary roadside LiDAR data are limited compared to the availability of autonomous driving datasets. Additionally, LiDAR data lacks portability of cameras due to the variability between manufacturers, number of beams, noise characteristics, and scanning patterns. This complicates research and commercial development, since the sensor used in the dataset collection should match the sensor used in the algorithm development and deployment.

For AV research there are many popular high-quality datasets such as NuScenes [24], Panoptic Nuscenes, [25] KITTI [26], Waymo [27], PandaSet [28], and A2D2 [29]. These datasets use sensors mounted on a moving vehicle, which produces very different data compared to a static sensor mounted greater than 5 meters from the ground. These differences include point cloud density distribution, object occlusion, background points, and mounting height. Thus, they provide limited utility for roadside traffic monitoring systems [18].

Recently, a few datasets have been released with static roadside LiDAR and camera data. Table 2.4 shows the details of the A9 [30], A9-I [31] LUMPI [32], DAIR-V2X [33], IPS300+ [34], and BAAI-VANJEE [35] datasets. Currently, only the A9-I dataset uses the same Ouster OS1 sensor available in the CMHT laboratory.

Table 2.4: Details for stationary roadside datasets.

Name	Year	LiDAR	Annotated Frames	Conditions	Scenario	Sensor Height
A9-Dataset	2022	Ouster OS1-64	5300 frames	Day, Snow, Fog	Highway, Urban	7 m
BAAI-VANJEE	2021	32L-LiDARp-R 32-beam	2500 frames	Day, Night, Cloud, Rain	Urban	4.5 m
DAIR-V2X_I *	2022	300-beam	10084 frames	Day, Night, Rain, Fog	Urban	-
IPS300+ **	2022	Robosense Ruby-Lite 80-beam	14198 frames	Day, Night	Urban	5.5 m
LUMPI	2022	VLP-16, HDL-64, Panda-64, PandaQT	145 minutes	Day, Cloud, Haze	Urban	-

* 40% of data from roadside perspective, 60% from vehicle perspective.

** Buses and trucks are sparsely represented.

2.8.2 CMHT Traffic Dataset

A custom dataset was collected for evaluating the system presented in this thesis. Three data collection locations were used to provide a realistic static scenario for traffic monitoring that a system would experience when deployed in the field. The rooftop location is approximately 12 meters above the road surface on the NE corner of the MARC building, facing Longwood Road South in the NNE direction. Figure 2.14a shows the data collection setup on the MARC rooftop, containing a laptop, sensors, and a tripod. The first roadside location is 3 meters from the road surface with a tripod on the rooftop of a sedan facing SSW on Longwood Road South. Figure 2.14b shows the first roadside location during nighttime recording. The second roadside location is on Frid Street, facing Longwood Road South. Table 2.5 provides an overview of the sequences that were labelled with 2D bounding boxes and track IDs. There are 6 sequences that cover day, dusk, and night lighting scenarios.

May10_R5 and **May10_R7** contain data from a sunny afternoon, with a medium-low traffic density. There is some slight occlusion from the streetlight, but overall these two sequences are considered easy.

Dec7_R1 was recorded in the evening during rush hour on a cloudy winter day. This sequence contains many detections due to the high traffic density, although the scene is darker, the lighting and quality is high. This sequence is considered easy.

Dec14_R1 was recorded at night, with a medium traffic density. Due to the low sensor height of 3 meters, vehicles may occlude each other, which gives this sequence a medium difficulty.

Oct18_R1 and **Oct_R9** contain nighttime data with a medium traffic density. Due to the low sensor height (3 meters) and multiple occlusions in the RoI, these sequences are hard.

(a) The data collection setup on the MARC rooftop, 12 meters above the road surface.

(b) The roadside data collection setup, 3 meters above the road surface.

Figure 2.14: The data recording locations (a) MARC rooftop, (b) roadside on Longwood Road South.

An ideal dataset would contain snow, rain, and other challenging conditions, however, the exposed nature of the laptop during data collection limited these opportunities. Furthermore, rooftop access to the MARC building is restricted to working hours and during fair weather.

Table 2.5: Overview of the CMHT Traffic Dataset.

Name	**Conditions**	**Mounting Position**	**Frame Count**	**Detection Count**
May10_R5	Day, sunny.	Rooftop (12 m)	186	370
May10_R7	Day, sunny.	Rooftop (12 m)	65	192
Dec7_R1	Dusk, cloudy.	Rooftop (12 m)	350	2071
Dec14_R1	Night, clear.	Roadside 2 (3 m)	233	663
Oct18_R1	Night, clear.	Roadside 1 (3 m)	200	633
Oct18_R9	Night, clear.	Roadside 1 (3 m)	115	348

2.9 Summary

This chapter covered current sensing technology for traffic monitoring and compared camera, RADAR, and LiDAR sensors. The details of the selected sensors are detailed and intrinsic and extrinsic calibration is explained. Finally, existing public datasets for traffic monitoring are reviewed and the CMHT Traffic dataset is presented.

Chapter 3

Camera Object Detection

3.1 Introduction

The image processing pipeline aims to detect and classify relevant objects in consecutive images from a video stream. This chapter will review image classification and object detection with supervised neural network techniques. And then describe and evaluate a camera-based detection pipeline.

3.2 Background

This section will introduce the background required for object detection in images.

3.2.1 Artificial Neural Networks

The artificial neural network (ANN) is loosely based on the idea of biological neurons. Equation 3.2.1 shows the output of an artificial neuron, which is represented by the sum of its inputs, each multiplied by a weight to represent the strength of the

connection with an additional constant. The output, z, is passed to a non-linear function known as the activation function. The term machine learning (ML) refers to the procedure of estimating the parameters in an ANN to perform a task [36]. Learning is enabled by backpropagation and an optimization algorithm.

$$z = \sum_i w_i x_i + b \tag{3.2.1}$$

A Feedforward Neural Network (FNN) is a type of ANN where input signals are propagated in one direction without cycles. Varying the number of neurons in a layer (width) and the number of layers (depth) of an ANN affects its accuracy for a specific task. Layers between the input and output are called hidden layers, as they can contain an arbitrary number of neurons. Since neurons are passed through an activation function, increasing the number of hidden layers affects the complexity of an ANN's decision boundaries. Figure 3.1 shows a simple feedforward ANN with 2 input neurons, a hidden layer with a width of 4, and 1 output neuron.

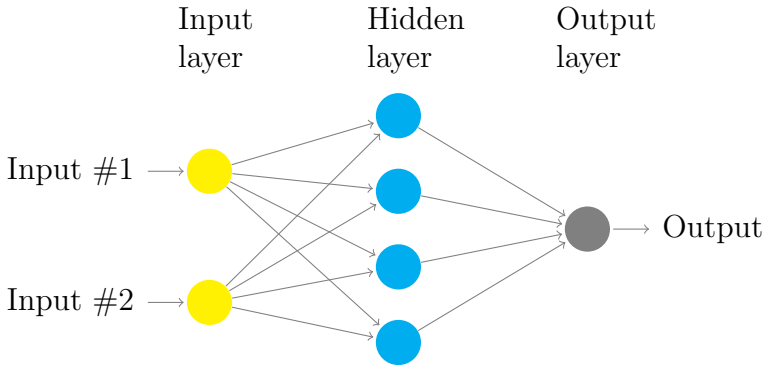


Figure 3.1: Feedforward Neural Network diagram.

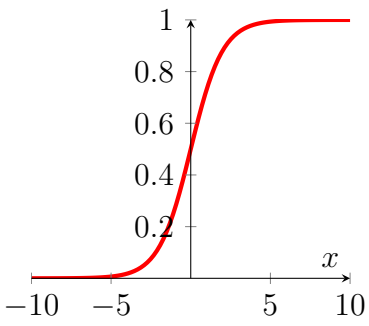
Activation Functions

Non-linear activation functions allow an ANN to produce non-linear outputs and can be categorized as saturated or non-saturated [37]. Saturated functions have bounded outputs, and non-saturated functions have non-finite bounds. The Sigmoid and hyperbolic tangent functions are popular saturated activation functions, shown in Figures 3.2a and 3.2b. Commonly used examples of non-saturated activation functions include the Rectified Linear Unit (ReLU) [38] and Leaky-Rectified Linear Unit [39]. Non-saturating functions are critical for increasing the learning efficiency of deeper networks. They aid backpropagation by maintaining a constant rate-of-change, even when input values are large, compared to saturating functions that have a diminishing slope for large values.

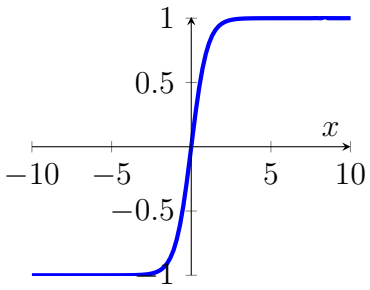
Backpropagation

Backpropagation is a method of computing the gradient of a loss function with respect to the parameters of a neural network. Hinton et al. popularized backpropagation for neural network training applications [40]. Computing the gradient of a loss function begins with a forward pass, or inference of the network with a training sample. Starting from the output layer, the loss is multiplied by the derivative of the previous layer and the activation function using the chain rule. This process is repeated up to the input layer.

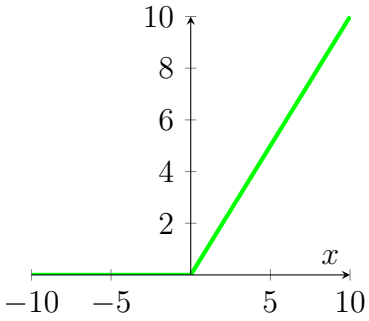
Once the gradient of the loss function is computed, it can be passed to an optimization function that determines the new values of each parameter in the network to reduce error for that training sample.



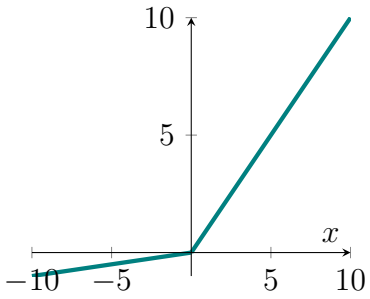
(a) Sigmoid function, $f(x) = \dfrac{1}{1+e^{-x}}$.



(b) Hyperbolic Tangent function, $f(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$.



(c) ReLU function, $f(x) = \max(0, x)$.



(d) Leaky-ReLU function, $f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$.

Figure 3.2: Common activation functions.

3.2.2 Evaluation Metrics

Metrics are critical to ML, since they are used in the loss function. Moreover, metrics are required to quantitatively compare multiple strategies for the same task. This section will cover some metrics for classification and visions tasks.

Precision and Recall

There are four possible results for a binary classification task: true positive (TP), true negative (TN), false positive (FP), and false negative (FN). Precision and recall

for a classifier are defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.2.2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.2.3}$$

where recall is the proportion of positive predictions and recall measures the proportion of correct predictions. There is a trade-off between precision and recall — increasing one can reduce the other. Plotting a precision-recall curve and estimating the area under the curve accounts for this tradeoff when comparing model performance.

Intersection over Union

The Intersection over Union (IoU) provides a measure of overlap between the predicted bounding box and the ground truth. IoU is also used in the loss function for training bounding box regression. Figure 3.3 shows the family of IoU metrics, including the Generalized Intersection over Union (GIoU) and Distance Intersection over Union (DIoU), which were proposed to speed up convergence and regression training [4].

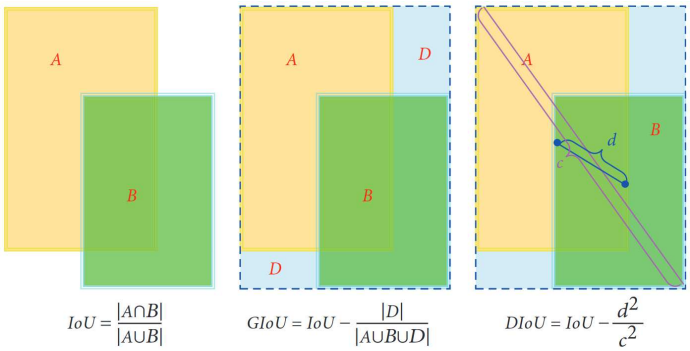


Figure 3.3: Diagram of IoU, DIoU, GIoU [4].

GIoU adds an area, D, to the calculation to eliminate the vanishing gradient problem when the boxes do not overlap, but it can still degenerate to IoU. DIoU adds the Euclidean distance of the centroids to the loss, which assists convergence by minimizing the centre distances directly.

Mean Average Precision

Mean Average Precision (mAP) is popular for evaluating object detection models. To calculate mAP, often called average precision (AP), an IoU threshold is applied to each detection to determine if it is a TP. Then the area under the precision-recall curve is calculated. The results for each class are averaged to determine the final mAP score. Some calculations use the average results from multiple IoU thresholds to evaluate the model at different levels of localization accuracy.

There are variations between mAP calculations for benchmark datasets. The Microsoft Common Objects in Context (COCO) [41] dataset uses 101 interpolated points along the precision-recall curve and averages multiple IoU values from 0.5 to 0.95 to compute AP. This calculation is more complex but provides fine-grained performance metrics. The KITTI dataset [26] first separates the evaluation dataset

into easy, moderate, and hard based on factors such as distance and occlusion. Then the mAP is calculated separately for the three groups using multiple IoU thresholds.

3.2.3 Non-Maximum Suppression

Non-Maximum Suppression (NMS) is a post-processing step for object detection that eliminates redundant detections with the goal of retaining a single bounding box per object. A common NMS algorithm method begins with the detection with the highest confidence score, then compares the IoU of the other detections. If the IoU is greater than a set threshold, that detection is removed. All detections that overlap with a more confident detection are removed. The performance of NMS relies on the IoU threshold and will vary depending on the needs of the application. A low threshold will increase the number of detections, but may increase the number of FPs. A high threshold may eliminate detections for similar objects that are very close together in an image.

3.2.4 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a subset of neural networks that employ kernels to map inputs into feature maps. Each kernel is learned during the training of the CNN, compared to traditional hand-crafted features. Equation 3.2.4 shows 2D discrete convolution given a $l \times l$ kernel, H. In computer vision applications, common kernel sizes are 3×3, 5×5 and 7×7. The convolution result is passed to an activation

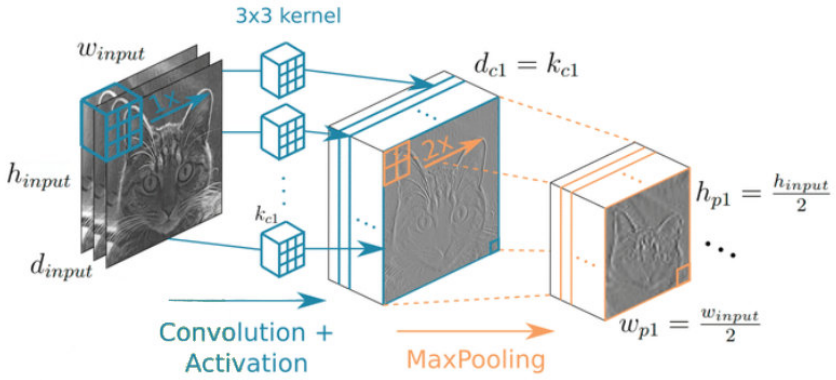
function.

$$y_{(i,j)} = \sum_{n=1}^{l} \sum_{m=1}^{l} H_{(n,m)} x_{(i+n,j+m)} \tag{3.2.4}$$

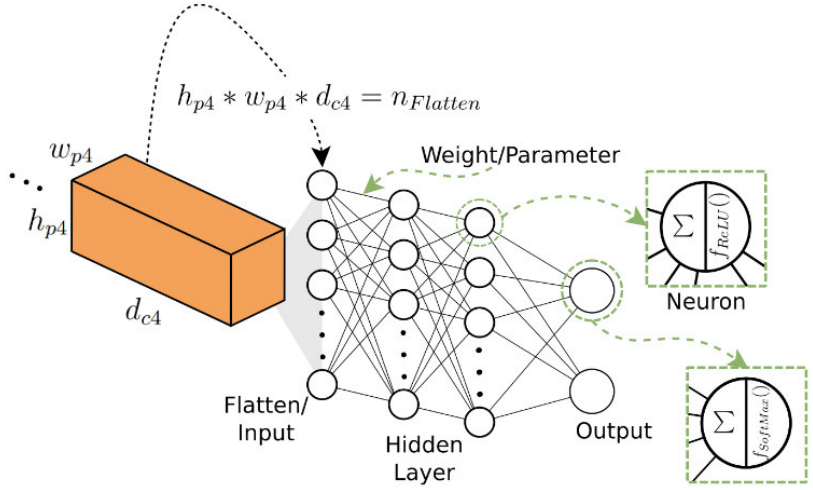
Figure 3.4a demonstrates the application of a convolutional layer, where image resolution is traded for feature map depth, d_{c1} which is proportional to the number of kernels used, k_{c1}.

To improve network efficiency, a technique known as pooling is employed to reduce layer sizes. Pooling layers downsample the information by retaining just one value from a kernel, usually the average or maximum value. A common pooling layer will retain the maximum value of a 2×2 kernel with a stride of 1. Figure 3.4a demonstrates the dimension reduction in layer when $w_{p1} = \frac{w_{c1}}{2}, h_{p1} = \frac{h_{c1}}{2}$. Max pooling functions do not have trainable parameters.

After a series of convolution and pooling layers, known as a convolutional backbone, the output of the final set of feature maps is flattened and attached to the classifier head, usually a fully connected ANN. Figure 3.4b shows the dimensions of the feature map dictating the input size of the ANN, $h_{p4} \times w_{p4} \times d_{c4} = n_{Flatten}$.



(a) Visualization of convolution, activation, and max pooling operations [42].



(b) Visualization of the convolutional backbone to ANN transition [42].

Figure 3.4: Details of a CNN architecture.

CNN structures are powerful because they learn hierarchical and spatial features from images and are robust to small changes in positions and distortion [43]. Many state-of-the-art image classifiers and object detectors employ a convolutional backbone in their network architecture.

3.2.5 Image Classification

Image classification is a computer vision task that takes an entire image and assigns a class. It is a building block for more complex tasks such as object detection. This section will highlight important image classification networks.

AlexNet

AlexNet [5] ignited a wave of research in deep learning for image classification in 2012 when it outperformed every method from the 2010 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [36]. AlexNet employed a new method of regularization, called dropout, in the fully connected layers to prevent over-learning. The authors showed that using a ReLU activation function speeds up CNN learning. Figure 3.5 demonstrates that ReLU reaches a 25 % training error approximately 30 epochs before tanh, which is a significant decrease. By combining a 63 % accuracy in the ILSVRC-2010 and reducing overfitting and training time, AlexNet popularized the ReLU function and deep learning for image classification.

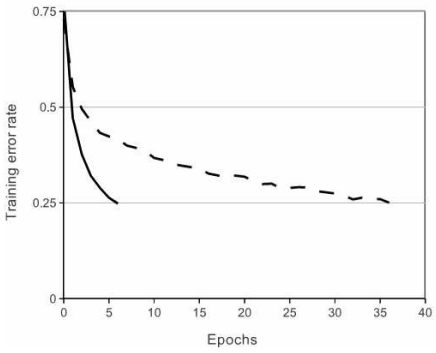


Figure 3.5: Training error versus time for ReLU (solid line) and tanh (dashed line) activation functions [5].

VGG

The VGG [44] CNN architecture builds off of AlexNet by increasing depth and selecting a 3×3 kernel compared to 11×11. This benefits the efficiency and training time of VGG over AlexNet by reducing the number of parameters by an order of magnitude. Two variations of the VGG architecture were released, containing 16 and 19 convolutional layers. Figure 3.6 shows the VGG16 architecture.

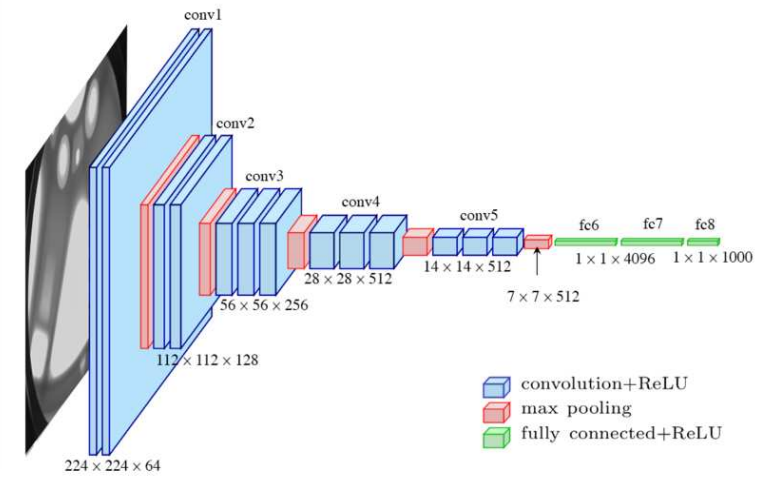


Figure 3.6: VGG16 architecture [6].

ResNet

The performance improvement between AlexNet and VGG networks may suggest that increasing the depth of a network will increase its performance, however, there is a limit. Figure 3.7 shows the training and test error of a 56-layer CNN is noticeably higher compared to a 20-layer variant. This effect is attributed to two issues, the vanishing/exploding problem [45] and accuracy degradation. First, the vanishing/exploding gradient problem is a disappearance or explosion of the variance of the

backpropagated gradient. This can occur in deep networks, which can potentially destabilize or slow training. However, this issue can be mitigated with normalized initialization [45]. Second, a degradation in output accuracy in larger networks can be seen in Figure 3.7, where errors for the deeper networks converge to a larger error.

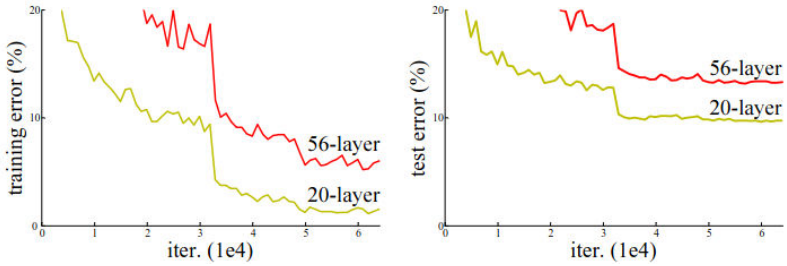


Figure 3.7: Training and Test error for 20-layer and 56-layer networks [7].

He et al. proposed the ResNet [7] architecture and a residual learning framework. Figure 3.8 shows the "shortcut-connections" or "skip-connections", which perform an identity mapping from one layer to another. The result of these skip-connections is faster convergence for model training and mitigating accuracy degradation in deeper networks.

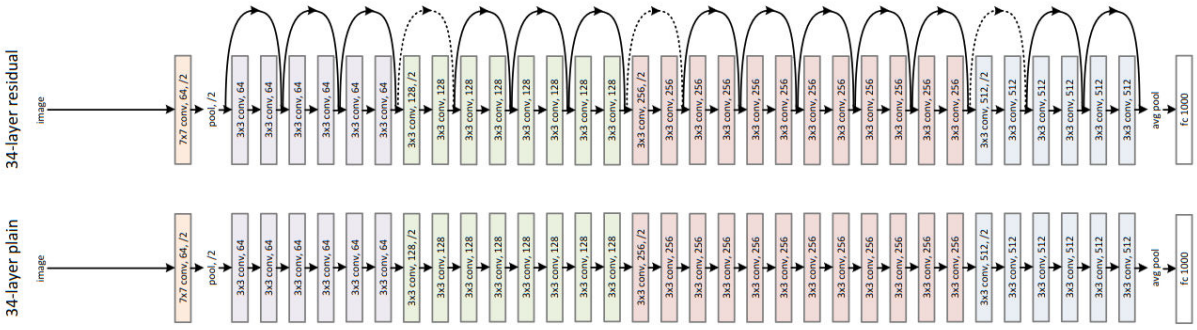


Figure 3.8: Example network architectures. Top: Residual network with skip-connections. Bottom: Plain network. [7].

3.2.6 2D Object Detection

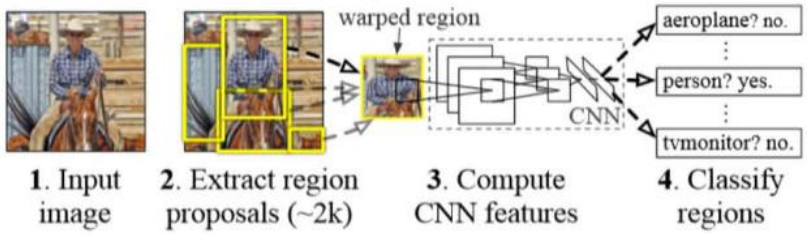
Object detection is a multitask problem, requiring the localization and classification of an arbitrary number of objects in an image. Traditional image processing methods use a sliding window and hand-crafted features for each position in the image. Then an ML classifier would determine if the current window contains an object of a specific class [46]. These traditional approaches have challenges when the number of classes grows. Deep learning is a popular approach to object detection.

Object detectors can be classified as one-stage or two-stage. One-stage detectors regress bounding boxes and classify detections in a single step. They are generally faster because of fewer parameters, and are more suitable for real-time systems. Two-stage detectors first create region proposals, then classify and regress bounding boxes. The following section will outline important networks from these two categories.

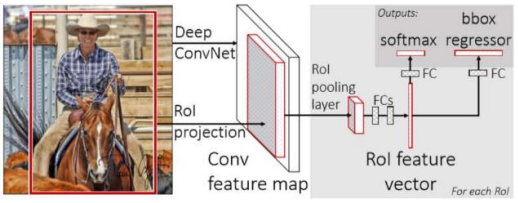
Two-stage Detectors

The Region-based CNN (R-CNN) family of networks includes R-CNN [47], Fast R-CNN [48], and Faster R-CNN [49], which are popular two-stage object detection models that successively improve inference and training speed. R-CNN uses a CNN for feature extraction for each region proposal generated via the selective search algorithm [50], then uses a traditional ML classifier. Fast R-CNN introduces a multitask head that combines classification and bounding box regression, which increases training efficiency [42]. Additionally, Fast R-CNN improves speed and performance by computing an image-wide shared feature map instead of recomputed features per proposal. However, Fast R-CNN still relies on the slow external selective search algorithm for region proposal. In 2015 Faster R-CNN introduces the Region Proposal

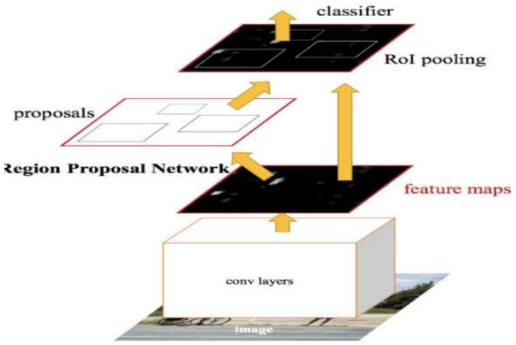
Network (RPN), which is a purpose built replaces the more general selective search algorithm. The RPN is a specialized network that re-uses the same feature map to generate region proposals. Faster R-CNN is end-to-end trainable, which is faster and more efficient than composing multiple components. Overall, Faster R-CNN is 245 times faster than R-CNN [51]. Figure 3.9 shows the evolution of the R-CNN family, where 3.9a is a composite of three independent components, compared to the two and one components found in 3.9b and 3.9c respectively.



(a) R-CNN diagram [47].



(b) Fast R-CNN diagram [48].

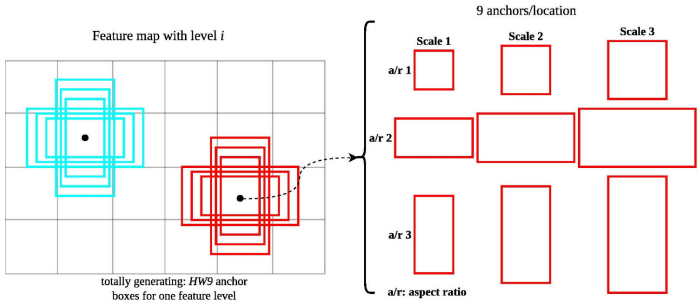


(c) Faster R-CNN diagram [49].

Figure 3.9: R-CNN family diagrams.

One-stage Detectors

One-stage detectors aim to achieve real-time and accurate results, by passing the input once through the network. Real-time refers to inference speeds of 20 FPS or greater.



MethodMethod

Figure 3.10: Example anchor box set [8].

Anchor boxes enable an object detection network to process an entire image at once. Instead of a sliding window approach found in two-stage detectors, anchor boxes for each class are spatially mapped from output to input. The anchor boxes are an *a priori* estimate of object locations, and through the training process, the network learns to adjust the size and location of the anchor boxes to make predictions. Anchor boxes should be designed to be similar to the scale and aspect ratio of the expected objects to increase accuracy. Figure 3.10 shows a set of anchor boxes that would be spatially distributed throughout a feature map [8]. Furthermore, anchor boxes are necessary for images because of the scale ambiguity of objects with different distances from the sensor [52].

The You Only Look Once (YOLO) family of networks are extremely popular for their speed, accuracy, and generalizability. The first YOLO network [53] was proposed in 2016, named **YOLOv1** in this thesis. To perform object detection in one-shot, the feature map produced by the 24-layer convolutional backbone is divided into a $S \times S$ grid. For each grid element, B bounding boxes and confidence scores are predicted per class. NMS is used to remove redundant detections. YOLOv1 uses a three-part loss function, consisting of localization, confidence, and classification losses. There are three primary drawbacks of this network. First, there is a limit of B objects of

each class that can be detection in each cell, thus multiple nearby objects could be ignored. Second, aspect ratios not found in the training data caused accuracy issues during testing. Finally, the network is not able to learn from fine-grained features since the detection occurs after the downsampled feature map is created [10].

YOLOv2 [54] presented several improvements over its predecessor, including a fully convolutional architecture, finer-grained features, and anchor boxes. YOLOv2 introduces the fully convolutional Darknet-19 backbone, which allows for a variable input size. The authors took advantage of this by changing the input size during training, increasing the model's robustness to different image sizes. YOLOv2 uses a set of anchor boxes in each grid element, to find good anchor box sizes k-means clustering was applied to the training data.

In 2018 **YOLOv3** [55] continued to improved performance. The backbone is replaced with Darknet-53, including 53 layers and ResNet [7] inspired connections. Furthermore, YOLOv3 adds multiscale prediction by branching three detection heads at different levels of resolution. Figure 3.11 shows the network's structure with three increasing prediction scales that help detect small objects, a weak point of YOLOv1 and YOLOv2.

A new set of authors introduced **YOLOv4** [56], which significantly improved accuracy and inference speed. This was accomplished with architectural upgrades and advanced training approaches. Figure 3.12 shows the YOLOv4 network architecture, which integrates multiple updates. The Darknet-53 backbone was modified with cross-stage partial connections (CSPNet) [57], which reduces memory consumption and computational bottlenecks. The red section in Figure 3.12 shows how CSPNet splits the base layer, passing one through a dense block, and connecting the other at

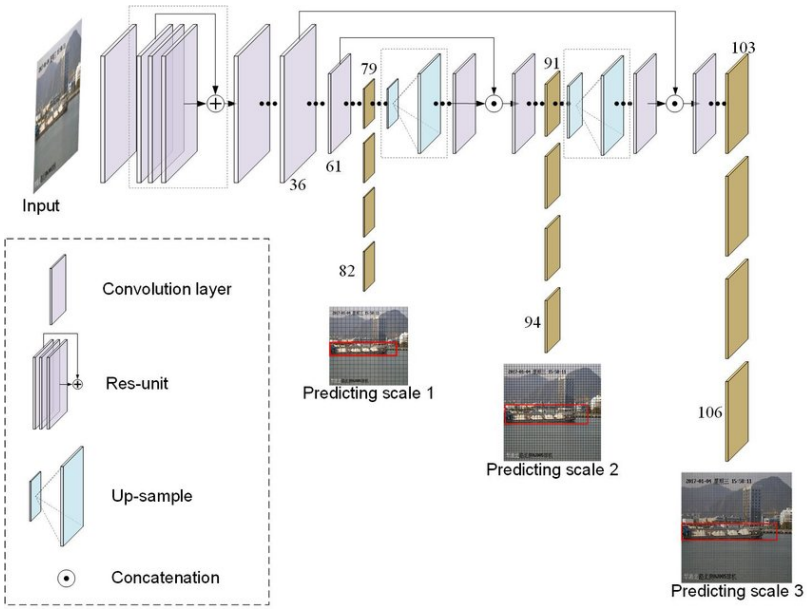


Figure 3.11: YOLOv3 network architecture. Darknet53 is branched in three places to provide multiscale detections [9].

the end. After the backbone comes the neck of the model, YOLOv4 uses spatial pyramid pooling (SPP) [58] combined with a modified path aggregation network (PANet) [59]. SPP extracts features from multiple scales to increase the receptive field, which increases detection performance [58]. PANet combines top-down, bottom-up and lateral connections. The bottom-up path provides the semantic information, while the top-down path upscales the feature maps to improve spatial resolution. Finally, the lateral connections concatenate the paths. YOLOv4 also leverages advanced training techniques such as mosaic data augmentation, which combines 4 training images into one, and the use of genetic algorithms for hyperparameter tuning.

YOLOv5 [15] was created as an independent repository, with no associated research paper. However, it takes advantage of improved strategies for data augmentation and training. YOLOv5 is developed with Python and PyTorch, which makes

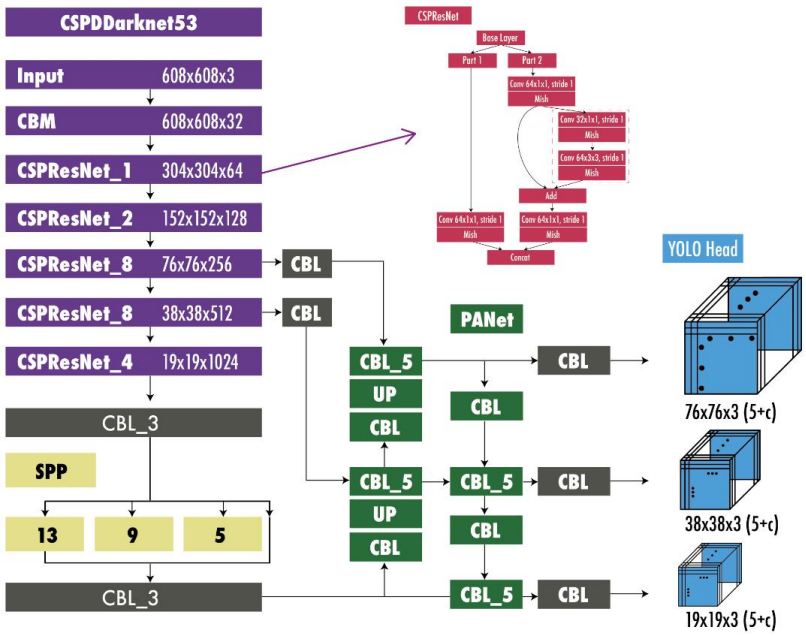


Figure 3.12: YOLOv4 network architecture [10].

it more accessible and simpler to integrate compared to Darknet's C++ implementation. The YOLOv5 object detector has multiple model sizes, from nano to extra-large. Larger models have more parameters and offer higher accuracy at the expense of inference time and memory.

3.3 Method

Since the focus of this thesis is the overall fusion pipeline, where camera-based object detection is one of many components, a pre-trained object detection network was leveraged. The pre-trained medium-sized YOLOv5 architecture was selected for its combination of accuracy and inference speed. Table 3.1 compares the different YOLOv5 weights based on size, accuracy, and inference time for a batch size of 1. The values $mAP_{50:95}$ and mAP_{50} represent the mAP for IoU values from $.5, 0.6, ...0.95$

and 0.5. Section 8 discusses the implementation and modularity of the system, which allows it to easily benefit from the state-of-the-art detectors.

Table 3.1: The speed, accuracy, and number of parameters for each YOLOv5 network given an input of 640 × 640 [15]. V100 refers to a specific GPU model.

Model	**size (pixels)**	**mAP**$_{50:95}$	**mAP**$_{50}$	**Speed CPU (ms)**	**Speed V100 (ms)**	**params (M)**
YOLOv5n	640	28.0	45.7	45	6.3	1.9
YOLOv5s	640	37.4	56.8	98	6.4	7.2
YOLOv5m	640	45.4	64.1	224	8.2	21.2
YOLOv5l	640	49.0	67.3	430	10.1	46.5
YOLOv5x	640	50.7	68.9	766	12.1	86.7

The camera-based object detection pipeline is simple. First, the raw image is scaled to 640 × 480 and the pixel values are normalized to $[0, 1]$. The preprocessed image is passed to the model for inference. Then, Non-Maximum Suppression (NMS) is applied to the detections to remove overlapping bounding boxes of the same object. The result is a set containing a 2D bounding box and class for each detection in the image.

3.4 Evaluation

Table 3.2 shows the mAP results for different classes at two IoU values. The first is the COCO default, using a range of IoUs from 0.5 to 0.95 and the second is using an IoU value of 0.5. The May sequences were recorded during the daytime and

YOLOv5 performs very well with mAP_{50}s of 0.92 and 0.95. For the Dec and Oct sequences, which are low light conditions, perform approximately 50% with mAP_{50} average of 0.45. The mAP decreases proportionally to the decrease in ambient light. Furthermore, the pedestrian detection results are significantly lower than the car class.

Table 3.2: mAP results for YOLOv5 on the CMHT Traffic dataset.

Sequence	**mAP**$_{50:95}$ All classes	**mAP**$_{50}$ All classes	**mAP**$_{50:95}$ Pedestrians	**mAP**$_{50}$ Pedestrians	**mAP**$_{50:95}$ Cars	**mAP**$_{50}$ Cars
May10_R5	0.72	0.92	-	-	0.78	0.92
May10_R7	0.83	0.95	-	-	0.72	0.92
Dec7_R1	0.38	0.83	-	-	0.26	0.78
Dec14_R1	0.36	0.45	0.015	0.024	0.27	0.37
Oct18_R1	0.30	0.47	0.18	0.28	0.41	0.66
Oct18_R9	0.29	0.45	0.002	0.013	0.58	0.90
Combined	0.48	0.68	0.066	0.16	0.50	0.76

Based on a qualitative evaluation of the detection pipeline, the YOLOv5 Medium sized model generalizes well to the CMHT Traffic dataset. During daytime conditions, most objects within 100 meters of the sensor are detected. Distant and highly occluded objects are missed, which is expected. Figure 3.13a shows a daytime detection example from sequence May10_R5. All the vehicles on the road are detected, including the vehicle partially occluded by the pole. During nighttime operation, image quality is reduced, subsequently diminishing range and accuracy. Figure 3.13b demonstrates a missed detection when occlusion and motion blur are combined in the

Oct18_R1 sequence.

(a) Example frame from sequence May10_R5. (b) Example frame from sequence Oct18_R1.

Figure 3.13: Example camera pipeline results.

A major drawback for the camera-based system is nighttime pedestrian detection. Figure 3.14 illustrates this problem, where the vehicles are assigned bounding boxes, but the pedestrian is missed. This is expected since YOLOv5 is trained on the COCO dataset, which does not have many images with very dark scenes and pedestrians at small scale.

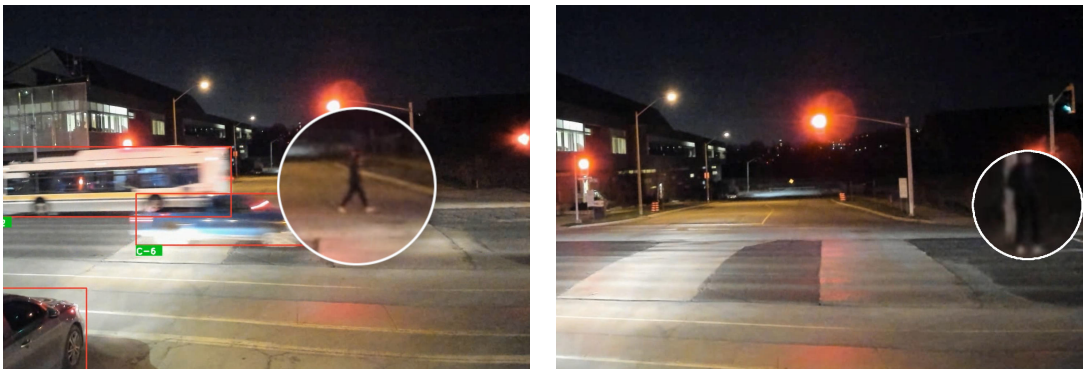


Figure 3.14: Examples of missing pedestrian detection at night.

3.5 Summary

This chapter covered the background, implementation, and evaluation for a camera-based object detection pipeline. First, ANN and CNN fundamentals, model evaluation, and relevant architectures for image classification and image-based object detection were reviewed. Then the implementation of the image-based object detection pipeline was discussed. Finally, the pipeline was evaluated qualitatively and with the COCO mAP metric.

Chapter 4

LiDAR Object Detection

4.1 Introduction

The purpose of an object detection algorithm is to locate objects in 3D space and provide the 3D size. The desired output is an accurate bounding box for each road user in the scene. Challenges for LiDAR object detection include the amount of data in each frame and the variation in density for objects at different distances from the sensor. This chapter will review traditional and data-driven approaches to LiDAR object detection, and then describe the algorithm implemented in this thesis. The results will be evaluated and discussed.

Object detection refers to the localization and classification of objects, however, this thesis will present LiDAR object classification in Chapter 5.

4.2 Background

Object detection algorithms for roadside LiDAR can be split into two categories, traditional (unsupervised) and data-driven (supervised). Figure 4.1 highlights the differences between the two methods. The supervised approach preprocesses the LiDAR frame and then uses a DNN to identify features and produce object detections in an end-to-end trainable network. For the unsupervised pipeline, background points are removed, and then the remaining foreground points are clustered to produce detections. The detections are then classified. The following sections will discuss these approaches.

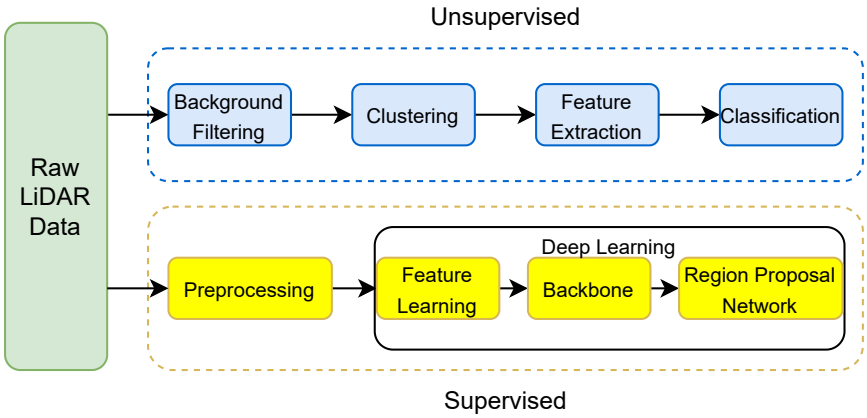


Figure 4.1: The flow of data in a roadside LiDAR object detection algorithm.

4.2.1 Traditional Methods

3D Object detection in point clouds for roadside applications is dominated by traditional unsupervised clustering approaches. These methods do not rely on large amounts of labelled data, which is difficult to find for roadside applications. Traditional techniques are well-studied and offer good performance, however, require expert

knowledge to tune parameters and may not generalize well to different deployment regions.

The general process includes processing the point cloud to remove background points and then clustering the remaining points, with the goal of each cluster representing an object of interest. The following subsections provide a detailed literature review of background subtraction and clustering methods in the context of traffic monitoring applications.

Background Removal

Background removal is the removal of irrelevant points from a point cloud. Existing methods can be categorized as referenced-based, voxel-based, and model-based methods.

In traffic monitoring applications the LiDAR sensor is fixed thus, without any road users present, each frame will contain the same background points. Many **reference-based** techniques borrow ideas from classical image processing to construct a background frame to subtract from the current frame. If the range of a point in the current frame is less than the background, it is assumed to belong to a foreground object and retained. Otherwise, the point is removed. Lee and Coifman [60] computed the background using a temporal median, setting the median of the range of each point observed over an extended time as the background. This assumes free-flowing traffic, as stationary objects become part of the background over time. For dense traffic scenarios, [60] use the mode of the furthest distance instead of the median. Zhang et al. [61] applied the Coarse-Fine Triangle Algorithm, which uses range

histograms for each point to automatically determine if a point belongs to the background. It requires the same assumption that the background is measured more often than a foreground object. The main drawback of reference-based methods is their dependence on an organized point cloud, which can limit preprocessing options, since not all algorithms retain cloud structure. Furthermore, any movement by the sensor, such as swaying in the wind, will cause misalignment in the current and background frames and reduce the accuracy of this method.

Voxel-based methods discretize point clouds into a 3D grid before determining if that cell belongs to the background [62]. Shackleton et al. [63] proposed an occupancy grid-based method, where each 3D cube maintains an occupancy history. The cubes that have been consistently occupied are considered part of the background. This approach trades the necessity of a structured point cloud with the burden of storing and maintaining a 3D occupancy map, which may be too computationally expensive for large outdoor environments. Wu et al. [64] introduced 3D-DSF, which discretized up to 1000 LiDAR frames into 5 cm cubes. The frames are spatially overlapped and concatenated, then cubes that surpassed the density threshold were considered part of the background.

Model-based methods remove points based on proximity to a defined plane or volume. Zimmer et al. [65] removed all points outside a pre-defined 3D RoI, then removed points within a threshold of a pre-defined ground plane.

Clustering

Different clustering algorithms will produce varying outputs given different data characteristics such as density, shape, and dimensionality. This section will cover some

popular clustering techniques for 3D data.

k-Means clustering separates n points with m dimensions into k clusters. Given data points $\mathbf{x}$, k-means aims to separate the data to minimize the within-cluster sum of squares (WCSS), as seen in Equation 4.2.1. Common implementations iteratively refine the point assignments to each cluster and apply a heuristic to reduce the runtime [66]. The primary drawback of k-means clustering for object detection is that the number of clusters needs to be provided, which is often not known in advance.

$$\arg\min_{x} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} ||\mathbf{x} - \mu_\mathbf{i}||^2 \tag{4.2.1}$$

Euclidean clustering groups all points that are within a set Euclidean distance of each other. Each cluster must contain a minimum number of points to be considered valid. The benefits of Euclidean clustering are that the number of clusters does not need to be specified, and the algorithm runtime is relatively fast.

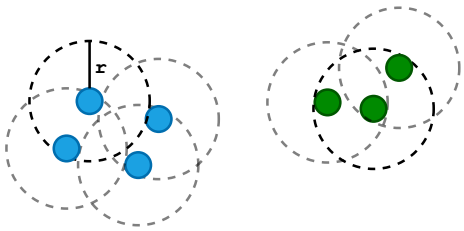


Figure 4.2: Euclidean clustering on a set of points with a distance threshold r.

Segmentation via **Region Growing** extends Euclidean clustering to compare additional features besides distance to add a point to a cluster [67]. Other features include surface normals or intensity, the additional data can help over-segmentation by ensuring points are not only close to each other but part of a similar surface or object.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [68] is a popular algorithm for 3D clustering due to its robustness against outliers. DBSCAN begins by computing point density. Points with a minimum amount of neighbours within a set radius are considered *core-points*. Clusters are formed by finding all reachable points from each *core-point*, non-reachable points are considered noise. DBSCAN has two parameters; the radius, *eps*, and the number of points within the radius to form a *core-point*, $minPts$. Traditionally the *eps* radius is the Euclidean distance, however, other distance metrics can be used. DBSCAN struggles with varying densities, which is common for LiDAR data, given the beam dispersion over long distances. Other disadvantages include sensitivity to parameter changes and the slower runtime compared to Euclidean and K-mean clustering methods. Figure 4.3 demonstrates how points outside the *eps* distance of any point in a cluster are considered noise, and given $minPts = 4$, two core points are located in this data.

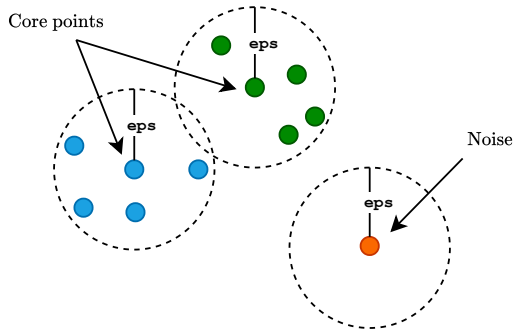


Figure 4.3: DBSCAN example, $minPts = 4$, *eps* is shown.

4.2.2 Data-based Methods

Data-based methods leverage deep learning to process point clouds to produce detections. These approaches require expensive GPU hardware, and require extensive

datasets with tedious 3D labels for training. Work such as [69] and [70] extended the CNN from 2D images to 3D point clouds, called volumetric CNNs. However, due to large computation time and inefficiencies from the sparsity of LIDAR data, these methods are not viable for real-time systems. More recently, new methods are split into two categories; voxel-based and point-based [52].

Complex YOLO [71] and PIXOR [72] are voxel-based methods that aggregate points into a 3D grid. Pseudo images are created by converting the vertical columns into a fixed-length handcrafted feature vector [52]. The pseudo images are processed by 2D image-based detectors with modified regression heads to obtain yaw angle and height. PointPillars [73] furthers this idea by creating pseudo images using PointNet [74] to replace the hand-crafted features, as well as ensuring the pseudo image is dense. These changes allowed PointPillars to achieve greater accuracy and speed compared to previous work. A drawback to voxel-based methods is the loss of information, since raw points are aggregated into the grid cells.

Point-based methods such as 3DSSD [75] and PointRCNN [76] use set abstraction layers for downsampling and feature extraction. The features are projected back to the points so that the 3D RPN can propose bounding boxes directly onto the points. The 3D detections are then refined. 3DSSD is a one-stage network, and PointRCNN is two-stage.

4.2.3 Evaluation Metrics

3D Object detection can be evaluated similarly to the methods described in Section 3.2.2 by extending IoU to 3 dimensions, using volume instead of area. Another method is to use 2D bird's-eye-view (BEV) projections to compute a 2D mAP, as seen in the

KITTI dataset [26]. Both methods require a labelled ground truth for the 3D LiDAR data, which is more labour-intensive to create compared to image data.

A simple approximation that avoids 3D ground truths is projecting 3D detections to the image plane. Then, 2D evaluation metrics can be used on labelled images. While this is easier than the 3D methods, it is less accurate because the image plane may have different levels of occlusion.

4.3 Method

The object detection module implemented in this thesis follows the procedure shown in Figure 4.4. Beginning with raw LiDAR point clouds, and outputting an oriented bounding box (OBB) for each object. The first steps focus on data reduction with downsampling, cropping, and ground point removal. Then, outliers are filtered before clustering the objects and computing the OBB.

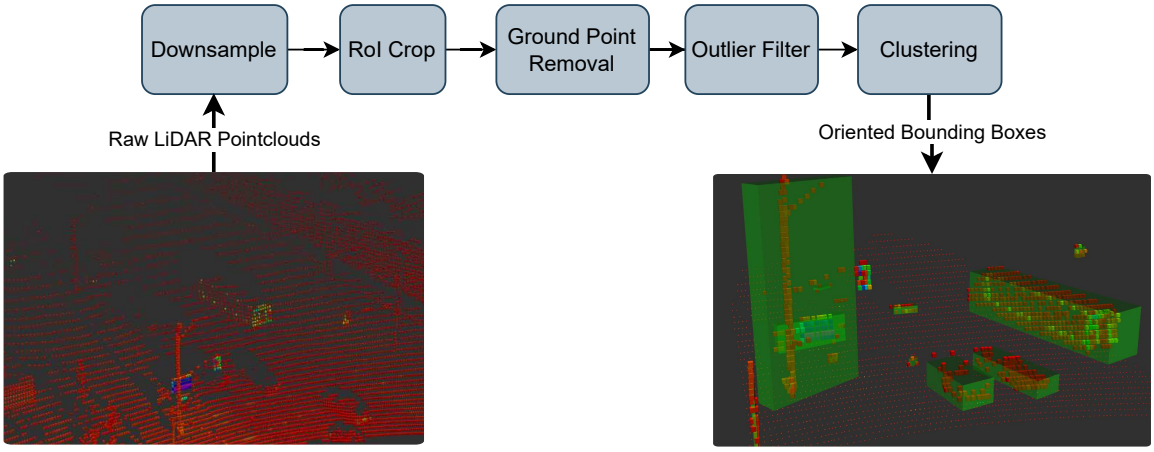


Figure 4.4: LiDAR object detection algorithm.

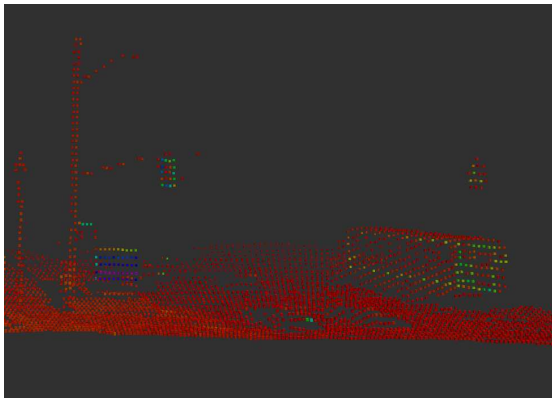
4.3.1 Data Reduction

Raw LiDAR frames arrive as a structured point cloud in Cartesian XYZ format, and each point in the 2048×64 array contains $[x, y, z]$ information. This amount of information makes processing each frame a computationally expensive task. Since many point cloud operations iterate over each point, reducing the number of points is effective for reducing the computation time.

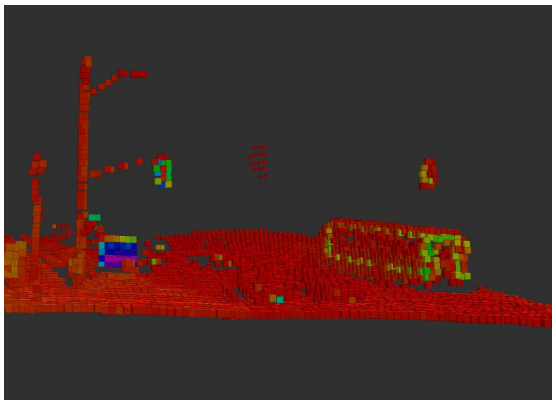
Voxel Grid Filter

Voxel grid filtering discretizes the 3D space into regularly spaced cubes called voxels (volume pixels). The points within each voxel are approximated by their centroid, effectively downsampling the point cloud. Figure 4.5 demonstrates the result of varying the size of the voxels. There is a tradeoff between the number of points and memory use versus the resolution of the point cloud.

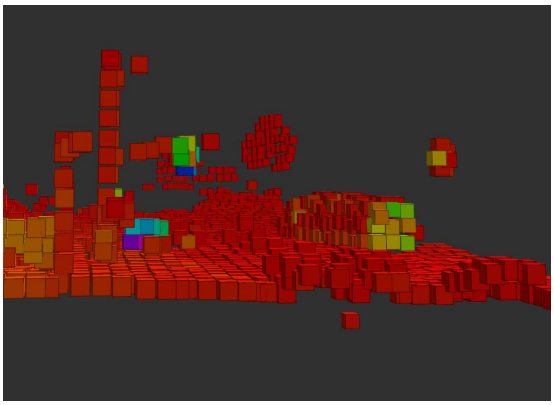
Another benefit to applying a voxel grid filter is that the effect of beam divergence is reduced. Since the density of measurements on an object decreases with distance, the voxel grid filter helps normalize the distribution of points throughout the point cloud [77].



(a) Voxel leaf size 0.1 meters.



(b) Voxel leaf size 0.3 meters.



(c) Voxel leaf size 0.9 meters.

Figure 4.5: Effects of voxel grid filter leaf size.

Region of Interest Cropping

Since this system is interested in traffic on roadways and intersections, a region of interest (RoI) can be identified in 3D space. Points which lay outside this region are removed. This is a simple method that reduces information that is not useful to the output of the pipeline. The RoI is determined with one or more 3D cuboid objects, defined by size (h, w, l) and pose relative to the sensor. The pose has a translation vector and orientation defined by a quaternion (x, y, z, w) relative to the ground reference frame. Figure 4.6 demonstrates an example of an intersection scene

with the RoI in the yellow prism.

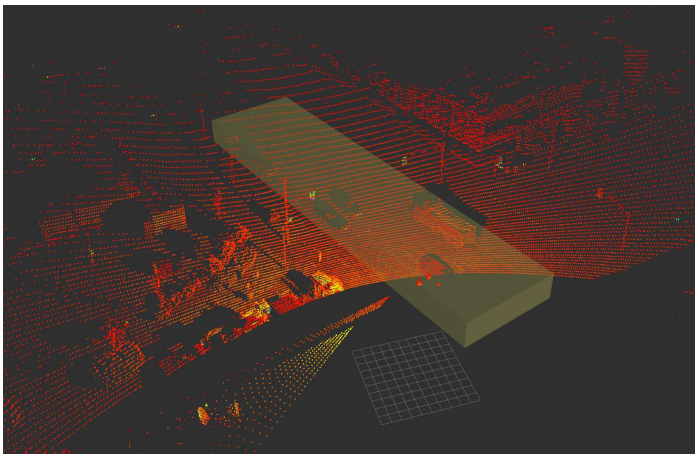


Figure 4.6: Visualization of the region of interest.

Ground Point Removal

Ground plane removal helps reduce the size of a point cloud and is important for downstream operations such as clustering by reducing object connectivity. Assuming the largest plane in the cloud is the road, a RANSAC-based approach [78] is used to robustly estimate the parameters of the ground plane, Algorithm 1 defines the process of fitting a plane model to the data. All the points within a set threshold of this plane are removed. Figure 4.7 shows the result of this procedure. Ground plane removal has the additional benefit of further reducing the size of the point cloud.

Algorithm 1 RANSAC

```
while n < N do
    Randomly select m points.
    Solve for the parameters of the model with the randomly selected points.
    Determine i, the number of points that are within the error tolerance, ε.
    if i/p > τ then
        Re-estimate model with all inliers.
        Exit while loop, save parameters.
    end if
    Increment n.
end while
```

Given n as current number of iterations, N is the maximum number of iterations, m is the minimum number of points required to estimate the desired model ($m = 3$ for a plane), p is the total number of points, i is the number of inliers for a given model and a given error threshold ϵ.

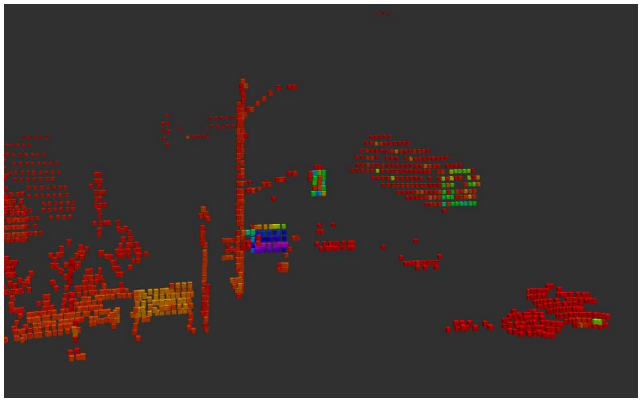


Figure 4.7: Roadside scene with ground points removed.

4.3.2 Outlier Removal

Noise and outliers in a point cloud can negatively affect the detection results by causing false detections or reducing the accuracy of detections. Furthermore, in conditions with rain or snow, the precipitation can reflect some LiDAR beams and produce speckled noise and points that are not useful for the algorithm.

Outlier removal is achieved by comparing each point's mean distance from n neighbours. If the mean distance is greater than a defined threshold, the point is removed. The threshold is a multiple of the standard deviation of the means and assumes that the distribution of points is Gaussian. Figure 4.8 shows an unfiltered point cloud on the left side, and the right side shows the filtered result. Removing outliers improves the accuracy of downstream processes and also reduces the number of points to process.

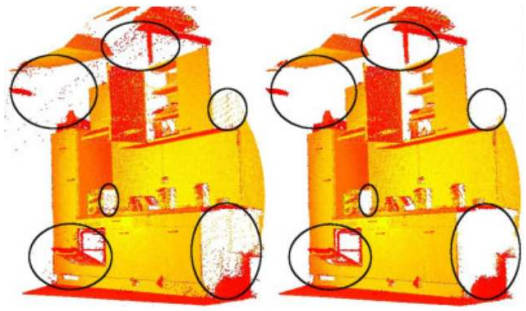


Figure 4.8: Example of outlier removal in a point cloud [11].

4.3.3 Clustering

The LiDAR pipeline uses Euclidean clustering because of its speed and performance. Algorithm 2 provides a high-level explanation of the clustering algorithm.

Algorithm 2 Euclidean Clustering [67]

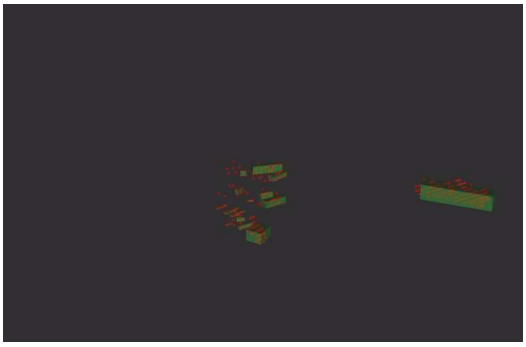
Create an empty list of clusters, C, and a queue of points to be checked, Q.
for $\mathbf{p_i} \in \mathcal{P}$ **do**
    Add $\mathbf{p_i}$ to Q.
    for $\mathbf{p_i} \in Q$ **do**
        Search for the set $\mathcal{P}_i^k$ of neighbours of $\mathbf{p_i}$ within radius, r.
        For each neighbour $\mathbf{p_i^k} \in \mathcal{P}_i^k$, if the point was not processed, add it to Q.
    end for
    Once all points in Q are processed, add Q to the list of clusters, C.
end for
Exit when all points $\mathbf{p_i} \in \mathcal{P}$ are processed and in a cluster within C.

Parameters were selected by iterating the radius, r, and the minimum number of points to validate clusters, $minPts$, until satisfactory qualitative results were achieved. Qualitative criteria include the following:

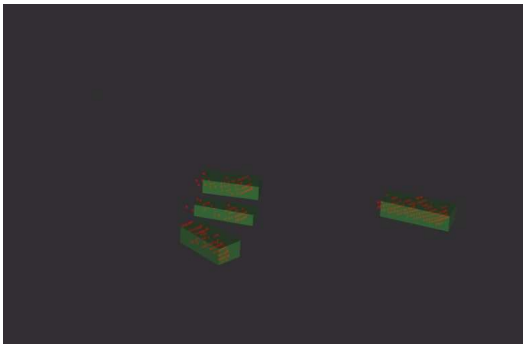
- Maximizing number of points in a cluster that belong to a single object.
- Reducing or eliminating clusters that contain points from more than one object.
- Size of the cluster as the object range increases (as points become sparse).
- Consistency of clustering at various distances from the sensor.
- Consistent performs across all sequences in the dataset.

Figure 4.9a shows the effects using $r = 0.3$m on the pre-processed LiDAR data, where r is not large enough to cluster all points belonging to a single object. Figure 4.9c uses $r = 3.3$m which causes over-segmentation by combining all three objects. Figure

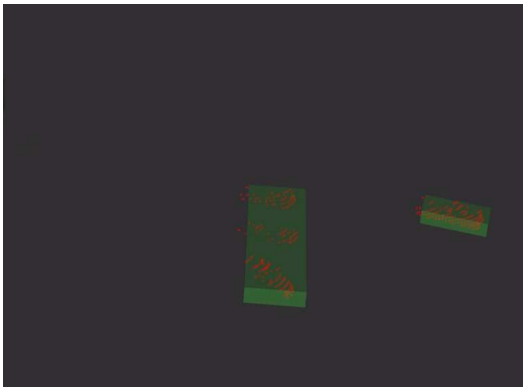
4.9b shows the result when $r = 1.3$m, each cluster contains the points of a single object. Figure 4.9d provides visual context for the processed LiDAR points, there are 4 vehicles in an intersection scene. The final values were chosen to be $r = 1.3$m, $minPts = 3$.



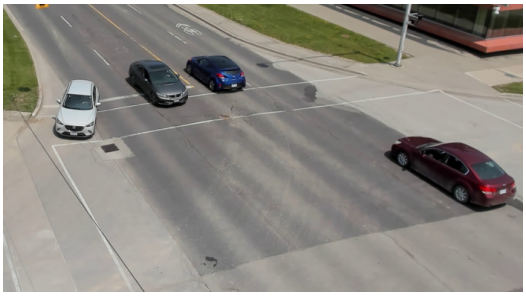
(a) Clusters with $r = 0.3$m. A single object is represented with multiple clusters.



(b) Clusters with $r = 1.3$m. Each object corresponds to a single cluster



(c) Clusters with $r = 3.3$m. Three objects are grouped.



(d) The synchronized image for the displayed LiDAR clusters.

Figure 4.9: Effect of varying the radius value, r, in a point cloud.

4.3.4 Bounding Box

A bounding box is a method of encoding the location of a detected object. This section details how a bounding box is computed.

Axis-Aligned Bounding Box

An axis-aligned bounding box (AABB) is the minimum bounding rectangle that has edges parallel to the axis of the frame of reference. Although simple to compute, each AABB is not guaranteed to be the minimum bounding volume and will have a heading of 0 radians. To compute the AABB, the minimum and maximum values of the cluster in x, y, z are obtained. The combinations of maximum and minimum values for each axis create the set of vertices of the AABB. For example, in a 2D cluster, the four corners are $(x_{min}, y_{min}), (x_{max}, y_{min}), (x_{max}, y_{min}), (x_{max}, y_{max})$. The AABB is common in 2D computer vision-based object detection algorithms.

Oriented Bounding Box

In 3D applications, the heading of an object is useful for tracking and downstream applications such as trajectory estimation. The oriented bounding box (OBB) is a rectangle with edges that are not parallel to the basis vectors of the coordinate frame. Algorithm 3 describes calculating the OBB in 3D space. The key to obtaining the OBB is principal component analysis (PCA). PCA represents an approximation of the original data with a set of orthogonal vectors, ordered by largest variance. In the traffic monitoring context, it is assumed that vehicles remain flat on the road and their orientation is described by position and yaw angle. Thus, using the first two principal components of a cluster is a good approximation of the vertical and horizontal axes of a vehicle. PCA is popular for computing the OBB because it balances the tradeoff between accuracy and computational speed.

Algorithm 3 Oriented Bounding Box Calculation

1: Determine principal axes using principal component analysis (PCA).
2: Determine a transform, T, from the origin frame to the principle axes using rotation between the axes and the centroid of the cluster.
3: Apply T to transform points to a space with the principal axes as the basis vectors and the centroid as the origin.
4: Determine $\min(x), \min(y), \min(z)$ and $\max(x), \max(y), \max(z)$.
5: Construct 8 vertices of the bounding box with a combination of the min and max points.
6: The heading is the yaw component of the T.
7: Return the size, centroid, and heading of the OBB.

Figure 4.10 highlights the difference in bounding boxes. The AABB (left) has more space, while the OBB (right) provides a more accurate object localization and heading angle. The LiDAR pipeline will use the OBB to localize 3D detections.

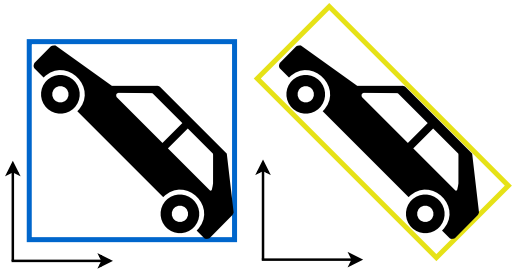


Figure 4.10: Comparison between the axis-aligned bounding box (left) and the oriented bounding box (right). The OBB provides a heading and contains less empty area.

4.4 Evaluation

Since labelling 3D point cloud data is labour-intensive, the 3D object detection algorithm was evaluated qualitatively. The preprocessing stages before clustering including, cropping, Ground removal, outlier removal, and bounding box calculations were visually validated during development.

To select a clustering algorithm, Euclidean clustering, region growing, DBSCAN, and OPTICS [79] clustering methods were applied to the data. All the algorithms performed similarly, thus, Euclidean was selected because of its faster runtime. The qualitative results suggest that the density of the point cloud has the largest effect on the quality and range of the detections. Occlusion cannot be compensated with a clustering algorithm. Furthermore, static elevated positions such as the MARC rooftop produce an ideal candidate for clustering, as each cluster that represents a vehicle is well separated from other road users. In intersections with many pedestrians and cyclists near each other, the clustering algorithm may need to be re-tuned to effectively separate individuals.

Figure 4.11a shows an example frame from sequence Oct18_R9, where the red points have been processed, and the green OBB are the detection results. The red and green circles show the detection results on occluded objects, in both the image and LiDAR frame. The LiDAR sensor easily detects the pedestrians regardless of the lighting conditions, shown by the yellow circle. Figure 4.11b is an example frame from sequence Dec7_R1. The vehicles inside the red and blue circles have very few LiDAR points. This sparsity reduces the detection accuracy and cannot be compensated for by a clustering algorithm.

(a) Example frame from sequence Oct18_R9.

(b) Example frame from sequence Dec7_R1.

Figure 4.11: Example LiDAR pipeline results.

4.5 Summary

This chapter covered LiDAR-based object detection. A review of traditional background removal and clustering is followed by data-based techniques. Then the object detection pipeline was proposed and detailed. Finally, the pipeline was evaluated and discussed.

Chapter 5

LiDAR Object Classification

5.1 Introduction

The goal of object classification is to determine the class of an object given a set of features. For traffic monitoring applications, an objects class is useful for collecting usage statistics on the type of road users interacting at a specific location. Once an object is given a class, it assumes the characteristics of that class, which can assist downstream applications such as tracking or trajectory prediction based on class-specific knowledge.

5.2 Background

This section will detail the classification algorithms and features relevant for LiDAR objects.

5.2.1 Classifiers

Supervised machine learning techniques are commonly used for classification problems. Once an object is localized via clustering, classification algorithms use information about each cluster to decide which category it belongs to.

Support Vector Machines

The Support Vector Machine (SVM) [80] is a supervised ML algorithm that can perform regression and classification tasks. For a classification task with N dimensional data, the SVM aims to find an $N-1$ dimensional hyperplane that maximizes the margin between two classes. Figure 5.1 shows a 2D example of binary classification with an SVM. The samples outlined in grey are the support vectors, which lie on the margins. Equation 5.2.1 defines the equation for the $N-1$ dimensional hyperplane, where $\mathbf{x}_i$ are the points and $\mathbf{w}$ are the weights that are learned during the training process. Equation 5.2.2 defines the output of the SVM, where y_i is the class label.

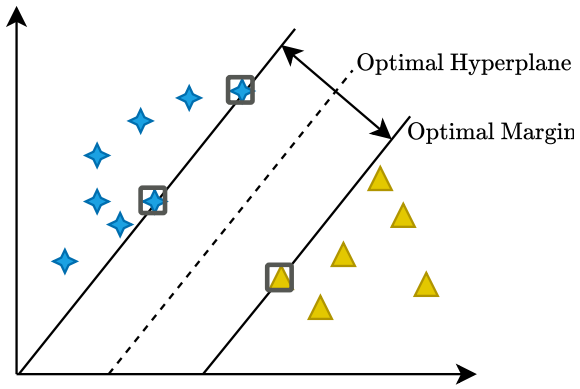


Figure 5.1: Binary classification with an SVM. The square points are the support vectors.

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{5.2.1}$$

$$y_i = \begin{cases} +1 & \mathbf{w} \cdot \mathbf{x} + b \geq 0 \\ -1 & \mathbf{w} \cdot \mathbf{x} + b < 0 \end{cases} \tag{5.2.2}$$

There are two types of margins that can be used: hard and soft. A hard margin requires every point from each class to lie on the correct side of the hyperplane, and cannot be computed if the data contains outliers. The soft-margin allows some points to violate the margin, which adds robustness to noise. Equation 5.2.3 shows the minimization goal of a soft-margin SVM, which uses the hinge loss to penalize points on the wrong side of the margin proportional to their distance from the margin. The parameter, λ, controls the trade-off between the size of the margin and the number of samples on the correct side of the margin.

$$\lambda ||\mathbf{w}||^2 + \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i - b)) \tag{5.2.3}$$

The SVM is capable of non-linear classification through non-linear kernels. The "kernel trick" implicitly maps data to a higher dimension that allows the SVM to perform non-linear classification. Figure 5.2 demonstrates this with 1D data that is mapped to a 2D plane, where a hyperplane can easily separate the two classes. Equation 5.2.4 and 5.2.5 show the polynomial and Radial Basis Function (RBF) kernels, where varying the kernel parameters controls the complexity of the decision boundary.

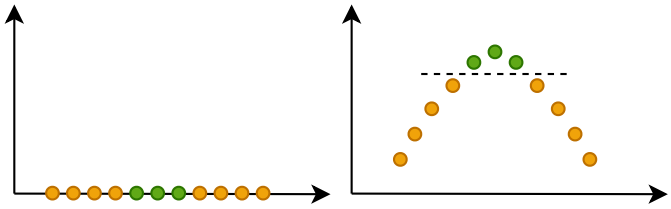


Figure 5.2: Example of non-linear mapping to allow linear separation.

$$K_{poly}(x_i, x_j) = (x_i \cdot x_j + c)^d \tag{5.2.4}$$

$$K_{RBF}(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right) \tag{5.2.5}$$

There are two strategies that enable multi-class classification with a binary classifier; one-vs-rest and one-vs-one. The **one-vs-rest** strategy trains one classifier per class to predict whether it belongs to that class or any other class. To make a prediction on a new sample, the classifier with the maximum score assigns the label. In **one-vs-one**, there is a binary classifier for each combination of individual classes. Given n classes, $n(n-1)/2$ binary classifiers will be trained. To predict the class of a new sample, each classifier is evaluated and the class with the most votes is assigned. One-vs-one requires more computational resources because it requires more binary classifiers. In both strategies, it is possible that certain samples will have an equal amount of votes for more than one class.

The SVM is memory efficient since it only requires the support vectors after training.

K-Nearest Neighbours

The K-Nearest Neighbours (k-NN) [81] classifier uses k closest points to determine the class of a test point. The k points are based on a distance metric such as Euclidean, Manhattan, or Minkowski. Figure 5.3 demonstrates classification with two different k values. For $k = 3$ the sample is assigned to the blue class, when $k = 5$ it is assigned to the yellow class. k-NN calculates each distance at runtime, which can be computationally inefficient depending on the dataset.

k-NN is sensitive to feature scale, thus normalizing the data and scaling features relative to their importance can help increase accuracy. Other methods such as dimension reduction, data-reduction and feature extraction are also important for increasing classification accuracy.

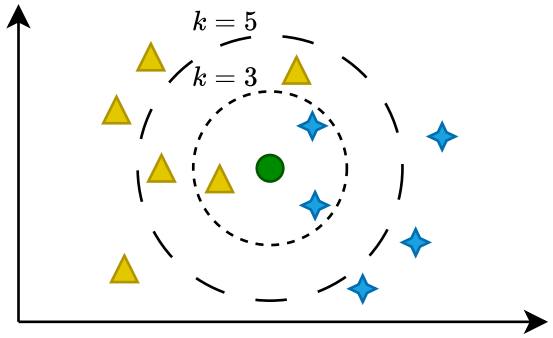


Figure 5.3: Example of k-NN classifications. The sample (green circle) is assigned to the blue class when $k = 3$. If $k = 5$ the sample is assigned to the yellow class.

The k-NN classifier is not memory efficient, since it keeps every training point in memory to compute the class of a test point. Furthermore, k-NN is sensitive to noise because it assumes similar objects are nearby in the feature space.

Decision Trees

A decision tree is a recursive structure that defines rules for classification or regression [82]. Each node contains a test that has a set of mutually exclusive outcomes, and each leaf has a class associated to it. To classify a sample, a path from root to leaf is traced depending on the result of each test. Decisions trees are easily interpreted since the tests and outcomes can be visualized in a flow chart.

Common decision tree algorithms use a greedy divide and conquer strategy to construct the tree. Tests are determined by selecting a variable that best splits the data at that node. Metrics to determine which variable to use include information gain, Gini impurity [83], and Chi-square.

Decision trees can be used in ensemble methods that combine multiple trees. Two ensemble methods are bagging and boosting. The bagging technique uses model averaging, with models that are trained on differently sampled datasets. The random forest classifier [84] is an example of the bagging ensemble method applied to decision trees, and benefits from reduced overfitting. Boosting sequentially combines multiple weaker classifiers to produce a result. AdaBoost [85] is a popular boosting algorithm that uses decision trees. Figure 5.4 highlights the parallel and sequential structure of bagging and boosting, respectively.

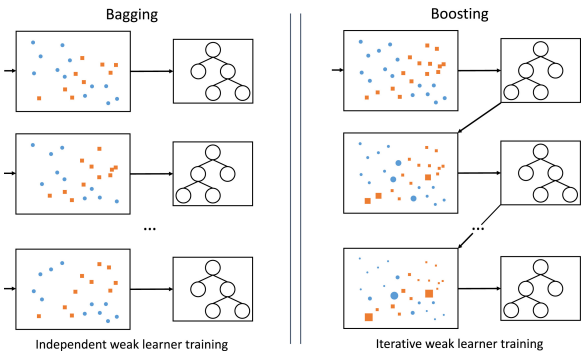


Figure 5.4: Comparison of bagging and boosting architecture [12].

5.2.2 Features

Traditional techniques for classification problems rely on handcrafted features by domain experts to accurately distinguish objects from different classes while maintaining intraclass similarity. Once features have been computed, an ML technique is applied to create decision boundaries in the feature space. There are many examples of classification for 3D point clouds, Lin et al. [86] propose an eight-dimensional solution that uses distance from the LiDAR, velocity, object size, number of LiDAR-measurement points, and distribution of reflection intensities with a multi-class SVM. Teichman et al. [87] uses twenty-nine cluster descriptors, including oriented bounding box size and multiple histogram of oriented gradients (HOG) [88] descriptors computed on 2D projections of the segment.

Data-driven techniques use learned features created by a DNN. A popular network for 3D point cloud classification is PointNet [74], which learns features directly from the unordered set of points. More details on LiDAR-based data-driven techniques are discussed in Section 4.2.2.

5.2.3 Evaluation Metrics

In addition to precision and recall introduced in Section 3.2.2, the F_1 score is used to compare classifier performance. Equation 5.2.6 defines the F_1 score as the harmonic mean of precision and recall.

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.2.6}$$

however, its use has been criticized for oversimplifying the analysis of classifiers, since not all applications benefit from equal weighting of precision and recall [89].

For multi-class problems, a confusion matrix provides insight into which classes the classifier performs well on and which classes get confused. Figure 5.5 shows how a confusion matrix shows correct and incorrect detections. A normalized confusion matrix converts the counts of TP, FN, FP, and TN such that each row sums to 1.

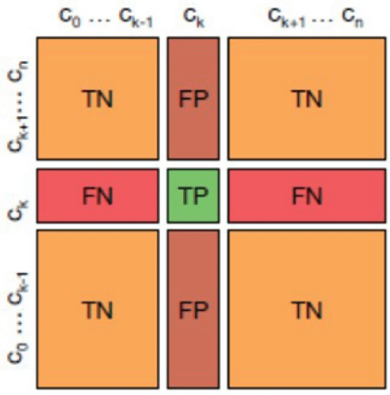


Figure 5.5: Visualization of the values in a confusion matrix [13].

5.3 Method

This section describes the methodology used to implement LiDAR object classification using supervised machine learning. First the dataset is described, then the features and classification are explained.

To create the training dataset, the labelled 3D ground truth from the R02_S01, R02_S02, R02_S03, R02_S04 sequence of the A9 dataset [30] were used to segment individual object point clouds from the scene. Features were computed for each object point cloud. In total, there are 9 classes with 42,373 detections, with 70% used for training and 30% for testing. Figure 5.6 shows the distribution of the detection classes, which is heavily skewed towards the car class.

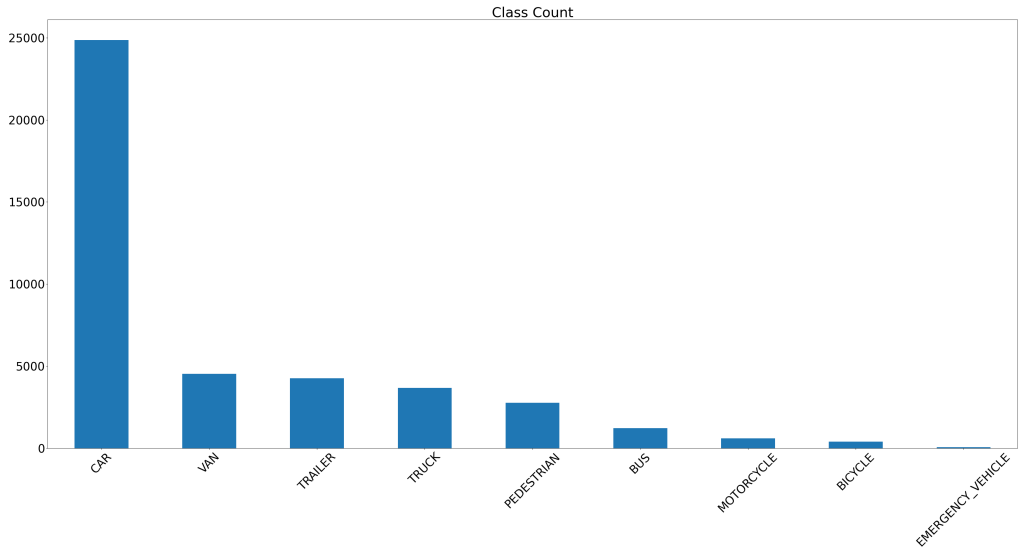


Figure 5.6: Plot of the class distribution of the training and test data.

The feature vector, $\mathbf{f}$, is a set of seven descriptors detailed in Equation 5.3.1.

Where density is the number of points divided by the OBB volume.

$$\mathbf{f} = \begin{bmatrix} \text{length} \\ \text{width} \\ \text{height} \\ \text{distance to sensor} \\ \text{number of points} \\ \text{point density} \end{bmatrix} \tag{5.3.1}$$

Once the feature vector is computed for each sample in the training set, it is scaled and reduced. Scaling the individual feature ensures that a single feature with a significantly larger magnitude does not bias the training result. It is assumed that the range of each vector in the testing data is similar to the training data, since the scaling values can not be changed by the test data. After scaling, the features are reduced by representing the data with the first three principal components determined by PCA. Three components were selected since using four did not increase accuracy, while 2 components had significantly lower results. The reduced feature vector decreases classifier training and inference time. Once trained, the scaling parameters, PCA parameters, and classifier parameters are saved to a file for processing and inferencing new data.

5.4 Evaluation

Accuracy and inference speed were used to determine which classifier to use. Table 5.1 shows the F1 score per class of a few popular classification algorithms and their

runtime averaged from 100 inferences of 50 feature vectors. The F1 score was macro averaged, since the class imbalance would grossly influence the results. The macro weight averages the F1-score of every class, irrespective of the size of the class. The Random Forest classifier was selected because it has the best combination F1 inference speed. It has the added benefit of interpretability since the decision trees can be converted to a human-readable format.

Table 5.1: Classifier results on test data.

Classifier	Macro-Averaged ↑ F1-score	Runtime (ms) ↓ (50 samples)
RBF SVM	**0.982**	4.11
Random Forest	<u>0.978</u>	<u>0.68</u>
Decision Tree	0.974	**0.04**
k-NN, $k = 7$	0.953	0.94
Linear SVM	0.768	4.27
AdaBoost	0.255	3.01

The Random Forest classifier has a macro-averaged F1-score of 0.978 and an inference time of 0.68 ms for 50 objects. Figure 5.7 shows the confusion matrix for the test split. The worst performing class is the Emergency Vehicle, it was often misclassified as the Van class. This is likely because there are only 22 occurrences in the entire dataset. However, every other class is $96\% - 100\%$ accurate. Although the performance on the test set is good, more data from other locations and weather conditions would help the classifier become more robust.

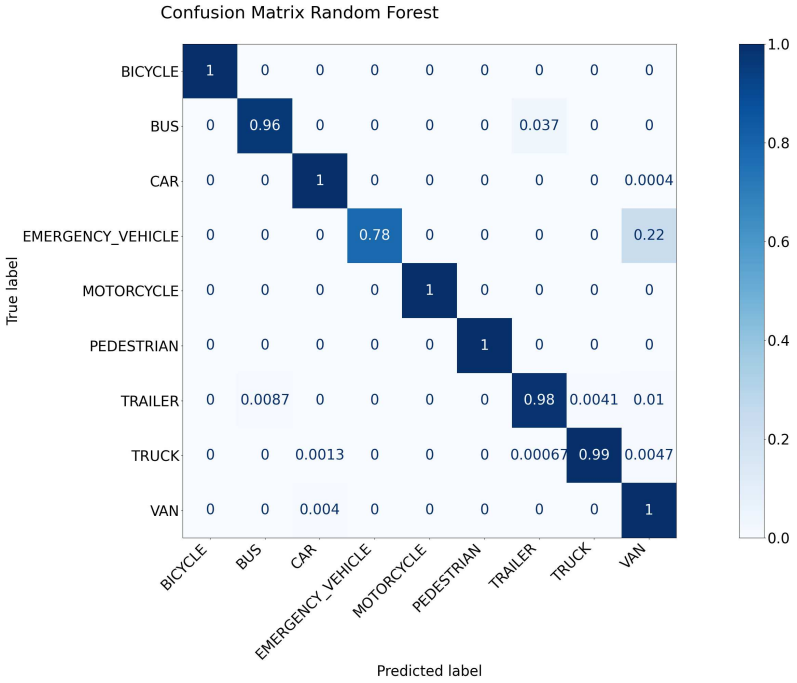


Figure 5.7: Confusion matrix of test results.

5.5 Summary

This chapter discussed LiDAR-based object classification. First, previous work and classifiers were reviewed. Then multiple classification algorithms are compared. The Random Forest classifier was selected, for its balance of accuracy and speed.

Chapter 6

Object Tracking

6.1 Introduction

The purpose of multiple object tracking (MOT) is to continuously maintain identities for detected objects, and provide trajectories given a sequence of frames. MOT algorithms can be split into online and offline categories. Online, or causal systems can use previous information up to the current frame, whereas offline algorithms can use information from every frame to compute results for the current frame. Another categorization is detection-based-tracking (DBT) and detection-free-tracking (DFT) [90]. DBT works on the principle of assigning detections to trajectories and is the most common since the detector provides a dynamic number of objects. DFT algorithms rely on initialization and then proceed to track the object over multiple frames without detection input. The primary drawback of DFT is manual initialization, which fixes the number of tracks. The remainder of this chapter will refer to DBT as tracking.

6.2 Background

6.2.1 Single Target Tracking

Accurately tracking an object using noisy sensor measurements and clutter is an important task. The Bayesian filter is the optimal tracking method, defined by a prediction step in Equation 6.2.1 and an update step in Equation 6.2.2. However, in practice, the Bayesian filter is often computationally intractable since it requires all previous observations.

$$p(\mathbf{x}_n|\mathbf{y}_{n-1:0}) = \int p(\mathbf{x}_n|\mathbf{x}_{n-1})p(\mathbf{x}_{n-1}|\mathbf{y}_{n-1:0})d\mathbf{x}_{n-1} \tag{6.2.1}$$

$$p(\mathbf{x}_n|\mathbf{y}_{n:0}) = \frac{p(\mathbf{y}_n|\mathbf{x}_n)p(\mathbf{x}_n|\mathbf{y}_{n-1:0})}{p(\mathbf{y}_n)|\mathbf{y}_{n-1:0}} \tag{6.2.2}$$

Assuming linear Gaussian systems, the Kalman Filter (KF) [91] provides an optimal estimate of the state, $\mathbf{x}$, given the following discrete state and output equations:

$$\mathbf{x}_n = \mathbf{A}\mathbf{x}_{n-1} + w_{n-1} \qquad w_{n-1} \sim \mathcal{N}(0, Q) \tag{6.2.3}$$

$$\mathbf{y}_n = \mathbf{C}_n\mathbf{x}_n + v_n \qquad v_n \sim \mathcal{N}(0, R) \tag{6.2.4}$$

where $\mathbf{A}$ is the state transition matrix, $\mathbf{C}$ is the output matrix, w_{n-1} is the process noise, and v_n is the measurement noise. The noise is assumed to be normally distributed with a zero mean, w_{n-1} can be adjusted to compensate for model inaccuracy. Figure 6.1 shows the predict-update cycle of the KF, where the variables with the $\sim$ are estimates and the $+$ denotes an *a priori* estimate. After incorporating the current measurement, the *a posteriori* estimate is produced. The Kalman gain, K_n, controls

how much the filter trusts the current measurement or predicted estimate.

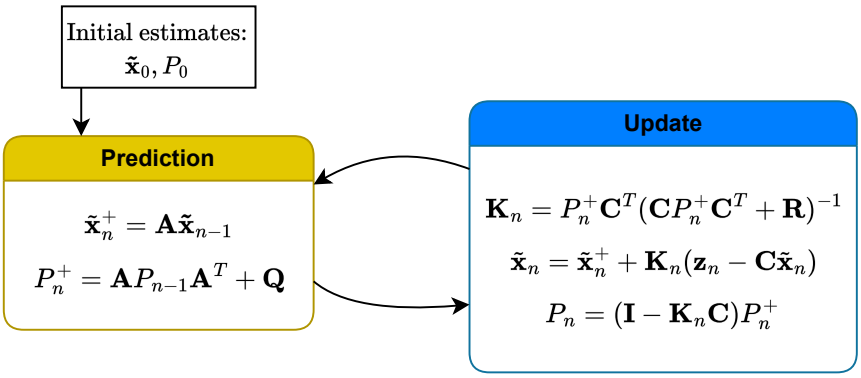


Figure 6.1: Kalman Filter diagram.

The Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) were proposed to handle non-linear dynamics by using first and third order Taylor series approximations however, they are more computationally expensive and still require the Gaussian assumption. The Particle Filter (PF) is another Bayesian-based state estimator that uses Monte Carlo simulations. PFs can approximate the posterior probability of non-linear and non-Gaussian systems with a set of N weighted points. This is achieved through sequential importance sampling and resampling of the points (particles). PFs require considerably higher computational resource.

6.2.2 Data Association

In the context of a MOT system, data association refers to maintaining the same object ID in subsequent frames. The measurements from the current time step need to be matched to the previous tracks.

The Hungarian Algorithm (HA) [92] performs one-to-one matching that solves the linear assignment problem. Given n workers and n tasks, the HA aims to assign one

task per worker that minimizes the total cost. A popular variant of the algorithm, known as the Jonker-Volgenant algorithm [93] has a time complexity of $O(n^3)$.

Probabilistic Data Association (PDA) takes a statistical approach to assigning a measurement to a single target. Instead of direct assignment, it maintains a probability of the measurement being related to the target. The Joint Probabilistic Data Association (JPDA) extends this concept to multiple targets. JPDA can suffer from track coalescence, where nearby tracks have a tendency to merge together.

6.2.3 Multi-Object Tracking

The simplest tracking algorithm in the 2D image space uses the HA to associate new and previous detections based on IoU scores. It is assumed that the current and previous detections which overlap the most are the same object. This naive algorithm fails when an object's inter-frame movement is large and consecutive detections do not overlap, such as fast-moving vehicles, lane changes, or accelerations.

The SORT algorithm [94], provides a simple and computationally efficient method of solving the MOT problem in the presence of good-quality detections. By directly tracking bounding box size and location in each frame, the KF provides an estimate for the position of the bounding box in each frame, while the HA associates detection with existing tracks. Unmatched tracks must be observed in three consecutive frames before being added to the track list. Tracks without new detections for five consecutive frames are deleted. SORT performs poorly when object motion is highly non-linear or objects are occluded. Extensions include OC-SORT [95], DeepSORT [96], StrongSORT [97]. OC-SORT uses detections to fix error accumulation from periods of occlusion with a virtual trajectory. DeepSORT and StrongSORT employ

learned object descriptors instead of handcrafted features such as IoU for data association. Additionally, StrongSORT uses a Re-identification (ReID) network to mitigate periods of occlusion by identifying the object as previously detected or not. ReID is an additional network that is inferenced with each detection, which adds computational time proportional to the number of tracked objects. The intention is that previously detected objects are assigned to their original track. Both deep descriptors and ReID networks must be trained.

6.2.4 Evaluation Metrics

Qualitative evaluations are a quick method to determine if detection or tracking functions are functioning as expected. However, to properly evaluate different methods, a quantitative metric is required. For MOT problems, quantitatively determining the best algorithm is non-trivial and is an active area of research.

Multi Object Tracking Accuracy and Precision

Multiple Object Tracking Accuracy (MOTA) and Multiple Object Tracking Precision (MOTP) [98] are used together for MOT evaluation. The MOTA score uses three tracking errors, FPs, FNs, and identity switches (IDSW). In the tracking context, an FP is an extra detection that is not mapped to the ground truth, and an FN is a missing detection. An IDSW is defined as a TP that swapped ID values with another TP, or a re-initialized track that has a different identity than before. Equation 6.2.5 defines the calculation of MOTA score, which averages the sum of all FN, FP, and

IDSW over the sum of ground truth detections. MOTA has a range of $(-\infty, 1]$.

$$\text{MOTA} = 1 - \frac{\sum(\text{FN} + \text{FP} + \text{IDSW})}{\sum \text{GT}} \tag{6.2.5}$$

MOTP is a measure for localization performance independent of maintaining consistent track IDs. Equation 6.2.6 defines MOTP as the average IoU score over all TP.

$$\text{MOTP} = \frac{\sum_{\text{TP}} \text{IoU}}{\sum \text{TP}} \tag{6.2.6}$$

Drawbacks to using MOTA is that it is biased towards detections. Furthermore, the impact of recall and precision are asymmetrical.

Higher Order Tracking Accuracy

The Higher Order Tracking Accuracy (HOTA) metric is composed of several sub-metrics that provide insight into a tracker's performance [99]. The sub-metrics are detection, assignment, and localization. The detection and assignment sub-metrics decompose further, each having a precision and recall metric. Equation 6.2.7 shows the computation for the overall HOTA score, where HOTA_α is the geometric mean of detection and association accuracy for a given localization threshold, α.

$$\text{HOTA} = \int_1^0 \text{HOTA}_\alpha \, d\alpha \approx \frac{1}{19} \sum_{\alpha \in \{\substack{0.05, 0.1, \ldots \\ 0.9, 0.95}\}} \text{HOTA}_\alpha \tag{6.2.7}$$

Integrating over the range of localization thresholds controls the tradeoff between association and detection, which overcomes the limitations of previous metrics that focused too much on either sub-metric.

A drawback of HOTA is that it does not penalize track fragmentation. Depending on the application, accounting for fragmentation could be more valuable than rewarding long-term global tracking [99].

6.3 Method

A modified-SORT algorithm is developed to track the 3D LiDAR detections. It remains lightweight but requires good-quality detections. It uses the *Jonker-Volgenant* algorithm [93] for data association, and the KF for tracking the state of each object. The tracker uses the 2D ground plane projection of the 3D detections. Equation 6.3.1 and 6.3.2 show the state vector, $\mathbf{x}$, and the measurement vector, $\mathbf{z}$.

$$\mathbf{x} = [x, y, l \times w, l/w, \theta, \dot{\theta}, \dot{x}, \dot{y}]^T \tag{6.3.1}$$

$$\mathbf{z} = [x, y, l \times w, l/w, \theta]^T \tag{6.3.2}$$

Where x, y are the position on the ground plane, l, w are the length and width of the OBB, and θ is the yaw angle of the OBB. A constant velocity model provides sufficient performance to predict and track between frames [100]. Equation 6.3.3 defines the constant velocity motion model used. Since the initial velocity is unknown, the velocity terms of the state vector are given a larger covariance than the position.

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} x_{n-1} + \dot{x}_n \Delta t \\ y_{n-1} + \dot{y}_n \Delta t \end{bmatrix} \tag{6.3.3}$$

Algorithm 4 details the modified-SORT algorithm. Where a minimum IoU cost is required for a match to be valid.

Algorithm 4 Modified SORT Algorithm

```
tracker_list = {}.
for detection in detection list do
    Take 2D ground plane OBB from 3D Detection.
    Extract the measurement, z, for data association.
end for
for Existing Trackers do
    Predict next state vector, x.
end for
Associate current detections and tracker predictions via IoU from rotated rectangles.
for matched detections do
    Retain camera detection, delete LiDAR detection.
    Update tracker with camera measurement, and increment hit counter.
    if hit counter > min_hits then
        Add tracker to tracker_list.
    end if
end for
for unmatched detections do
    Create new tracker and set hit counter to 0.
end for
```

The height of each object is set to a static height based on the class.

6.4 Evaluation

This section will evaluate camera-only and LiDAR-only MOT results with the HOTA metric on the CMHT Traffic dataset. Since only 2D images have been labelled, the 3D LiDAR tracking results have been projected to the 2D image plane for evaluation. Detections from the ground truth, camera, and LiDAR projections are preprocessed by removing detection results outside the RoI.

Camera-only Tracking

To evaluate the camera-based approach on the CMHT Traffic dataset, the camera detections are passed to the SORT tracker to generate the MOT output. Table 6.1 shows the HOTA score, sub-metric accuracies, and number of detections and IDs. The camera pipeline performs better on sequences with more ambient light, and significantly degrades in the darker sequences. These results are expected for a camera-based system. First, less ambient light causes the camera to compensate with a slower shutter speed, which creates motion blur for fast moving objects. Second, dark-coloured objects are difficult to distinguish from the background, especially for pedestrians on the sidewalk. Detections are missed by the object detection model because it struggles to generalize to the distorted and dark images. Figure 3.14 demonstrates missing pedestrian detection problem. This is shown by the degraded detection accuracy score and lower detection counts. However, localization accuracy remains mostly unaffected since the objects that are detected have accurate bounding boxes.

Table 6.1: Tracking performance of camera pipeline.

Sequence	HOTA ↑	DetA ↑	AssA ↑	LocA ↑	Dets	GT Dets	IDs	GT IDs
May10_R5	65.5	70.6	61.3	85.6	337	370	20	12
May10_R7	**72.7**	77.0	69.4	86.9	190	192	12	8
Dec7_R1	64.8	68.2	62.0	85.2	1810	2071	53	39
Dec14_R1	51.0	44.7	59.7	83.8	396	663	23	23
Oct18_R1	**31.3**	30.4	32.2	79.7	445	633	38	20
Oct18_R9	32.5	26.9	39.3	80.5	127	348	7	6
Combined	56.8	55.6	58.9	84.5	3305	4277	153	108

LiDAR-only Tracking

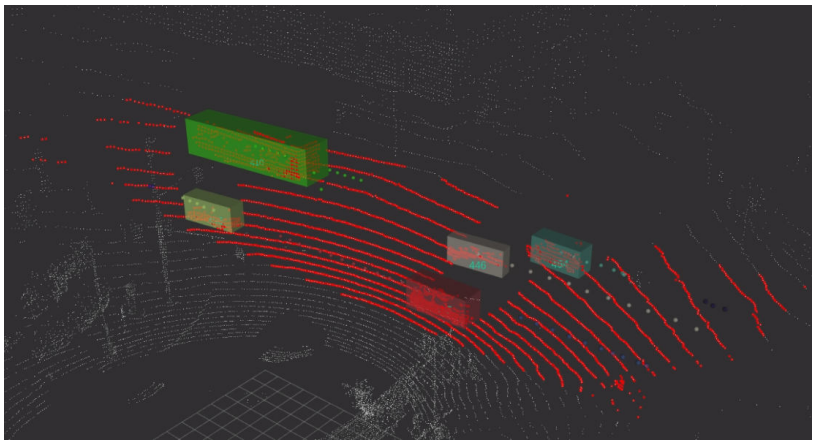
To evaluate the LiDAR-based approach on the CMHT Traffic dataset, the 3D LiDAR detections are passed to the modified-SORT tracker and then projected to the 2D image plane. Table 6.2 shows that the LiDAR results follow the same trend as the camera pipeline, but performs worse overall. There are three primary reasons why the LiDAR is score significantly lower than the camera score; 2D projection error, data density, and sensor range. First, the 3D perspective causes the projected bounding boxes to be larger than the 2D ground truth, which impacts the localization score. Second, since the LiDAR data is less dense, some object may be represented by only three points. The resulting OBB will be much smaller than the camera result, which reduces the IoU score. Finally, the camera detections are more consistent within the working range, where the LiDAR point cloud becomes less dense after 40 meters, which causes inaccurate bounding box estimations. Figure 6.2 shows example tracked OBBs with track IDs and coloured spheres to represent previous centroid location per

object.

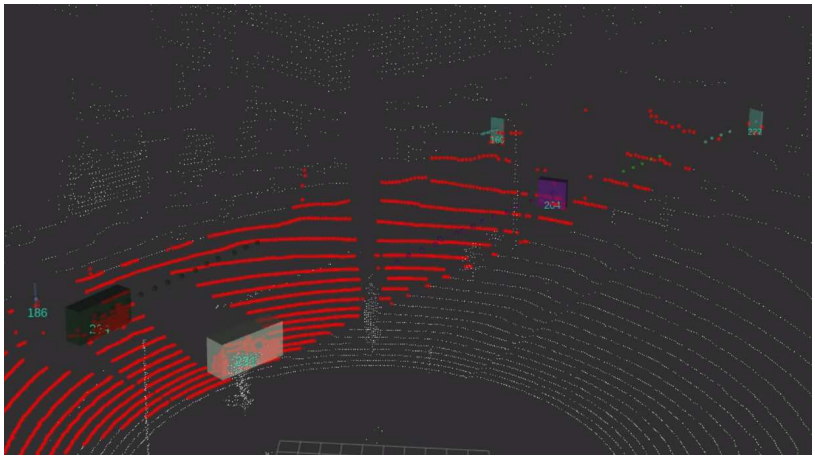
Table 6.2: Tracking performance of the projected LiDAR pipeline.

Sequence	**HOTA ↑**	**DetA ↑**	**AssA ↑**	**LocA ↑**	**Dets**	**GT Dets**	**IDs**	**GT IDs**
May10_R5	33.8	26.8	44.4	66.2	238	370	16	12
May10_R7	**38.7**	33.0	48.8	67.9	153	192	8	8
Dec7_R1	31.5	29.0	35.1	66.1	1515	2071	50	39
Dec14_R1	35.2	28.9	44.3	66.8	469	663	23	23
Oct18_R1	**26.0**	22.8	30.3	65.7	378	633	27	20
Oct18_R9	30.5	23.3	40.4	65.6	206	348	6	6
Combined	31.9	27.6	38.3	65.9	2959	4277	130	108

However, the LiDAR outperforms the camera in pedestrian detection. The lighting conditions do not have a noticeable impact on the detections, this is exemplified when comparing camera detections from Figure 3.14 and LiDAR detections from Figure 4.11a.



(a) Example detection result from sequence Dec14_R1.



(b) Example detection result from sequence Oct18_9.

Figure 6.2: 3D LiDAR tracking detection results.

6.5 Summary

This chapter covers multi-object tracking. A review of single-target trackers and data association, and previous works are presented. Lidar-only and camera-only tracking pipelines are evaluated using the HOTA metric on the CMHT Traffic dataset. Both modalities have similar trends, the best and worst scores for each pipeline occurs on the same data. The camera pipeline has a higher overall score, since the projections

are less accurate in the 2D plane. Chapter 7 will cover the fusion of camera and LiDAR sensors, and Chapter 9 provides an analysis over different IoU thresholds for the camera, LiDAR and fusion pipelines.

Chapter 7

Sensor Fusion

7.1 Introduction

Sensor fusion, or information fusion, is a technique of combining multiple data sources to accomplish a task more effectively compared to a single sensor. Sensor fusion aims to leverage the strengths of each sensing modality to overcome an individual weakness. In the context of traffic monitoring, challenges include small object detection, low light, and changing weather conditions. Sensor fusion can help overcome missed detections and increase the accuracy of a system. This section will provide background on sensor fusion for object detection, and then detail the method and evaluation of the camera-LiDAR fusion pipeline.

7.2 Background

Sensor fusion techniques can be split into three categories: early fusion, late fusion, and hybrid fusion. In early fusion, data-level or feature-level information from each

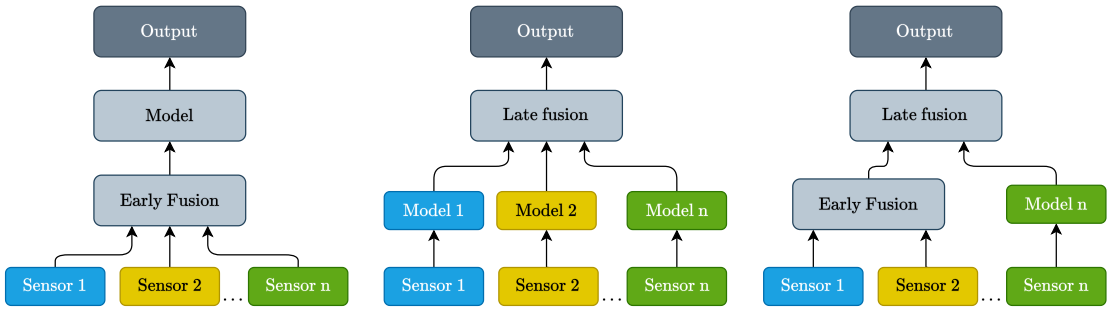
sensor is fused before entering the detection model. Directly using the data or feature vectors retains the most information and allows a model to learn the relationship between the sensors [101]. In BEVFusion [102] camera and LiDAR data are fused in the top-down BEV space. The network extracts image features and projects them to the BEV and then concatenates this data with the LiDAR features. The fused information is used for both object detection and segmentation tasks. Other examples of early fusion with camera and LiDAR include [103], and [104]. A benefit to early fusion is that the training process can be simplified with a unified end-to-end model, which makes the training procedure more efficient. Figure 7.1a shows the early fusion framework, which combines multiple data sources into a single model. A drawback of early fusion is the amount of data required to train the model, since combining heterogeneous sources often requires a larger, more complex network with higher dimensional inputs. Moreover, a single sensor failure can stop the entire network.

Another application of early fusion is fusing homogenous sensor data. Data from the same type of sensor at a different orientation can be fused at the data level to increase FoV or data density. By stitching images and point clouds from multiple views, Zimmer et al. [65] provides the opportunity for longer continuous tracking results and a greater number of detection compared to one set of sensors.

In late fusion, decision-level information such as detection or track results from multiple sensors are combined. Figure 7.1b shows the framework for late fusion; a model for each sensor modality is trained and then integrated with the other models. A benefit to late fusion is that an error with one sensor is likely independent of the others, and the system can produce output in the event of a sensor failure. Moreover, each model is simpler and easier to train, but, requires separate training

procedures and is no longer end-to-end. Drawbacks to late fusion include a potential loss of information, and a requirement for multiple concurrent models. Senel et al. [105]propose a late-fusion method that supports an arbitrary number of sensors and can operate both on a vehicle or roadside. Detections are assigned via the Hungarian Algorithm and tracked via the Unscented Kalman Filter. In [106], a camera-LiDAR fusion model is proposed that combines projected 3D LiDAR detections and 2D camera detections using a learned fusion model. Furthermore, a novel appearance and motion-based association metric is proposed. In [65], camera and LiDAR detections are combined after extracting 3D detections from each modality. Other examples of late fusion include [107] and [108].

Hybrid fusion combines early and late fusion techniques, shown in Figure 7.1c. While it has the benefits of both frameworks, it is the most complicated approach since it requires both uni-modal models and a multi-modal early fusion model to be trained.



(a) Early fusion framework. (b) Late fusion framework. (c) Hybrid fusion framework.

Figure 7.1: The three sensor fusion frameworks.

7.3 Method

In this thesis, a late fusion method between camera detections and projected 2D LiDAR detections is proposed. This pipeline can be used in two ways, 2D mode and 3D mode. 2D mode leverages detection contributions from both sensors, and the fused track outputs are in the 2D image plane. This combines the long distance and high quality daytime detections of the camera with the nigh-time performance of the LiDAR using a simple matching based strategy.

In 3D mode, the output position and velocity of the tracks are in 3D. This fuses the 3D position and velocity from the LiDAR with the camera's classification information. Figure 7.2 illustrates the flow of information, beginning with detections from each sensor, then association and tracking to produce the fused IDs and bounding boxes. A benefit of this approach is that RADAR or other sensor data can contribute to this strategy without significant modification of the fusion module, and without updating of the camera and LiDAR detectors.

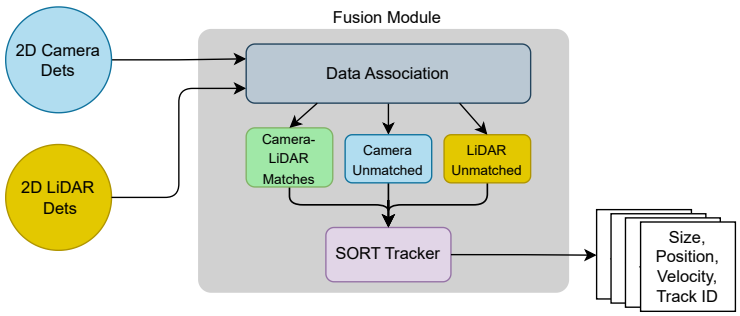


Figure 7.2: Fusion module diagram.

To fuse camera and LiDAR data, the 3D LiDAR OBBs are projected to the 2D image plane so that both modalities are in the same space. A 3D point in the LiDAR frame can be projected to the image plane using the intrinsic matrix, K from equation

2.7.1, and the extrinsic transformation matrix, ${}^{L}T_{C}$ from equation 2.7.5, as shown

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = {}^{L}T_{C} \begin{bmatrix} X_L & Y_L & Z_L & 1 \end{bmatrix}^{T} \tag{7.3.1}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix} \tag{7.3.2}$$

where u, v are the corresponding pixel coordinates.

To convert LiDAR detection into 2D, the 8 corners of the 3D bounding box are projected into the image plane. Then the bounding box of the corner points is computed by finding the AABB shown in Section 4.3.4.

Once the 2D LiDAR bounding boxes are obtained, the synchronized camera and LiDAR detections are fused. Algorithm 5 shows the steps to fuse the data and maintain the track IDs. An IoU-based matching algorithm matches detections from the camera and the LiDAR sensor, if they surpass a set threshold. Once matched, the 2D data and class are retained from the camera data and the 3D location and velocity are kept from the LiDAR detection. The fused list is the set of matched camera detections, unmatched camera detections, and unmatched LiDAR detections. The fused list is tracked using the SORT algorithm which produces the final set of bounding boxes, classification IDs, and track IDs.

Algorithm 5 Fusion Algorithm

$f\_list = \{\}$
Initialize SORT tracker.
while Synchronized camera and LiDAR data arrive **do**
    Perform data association between camera and LiDAR AABBs using IoU.
    for matched detections **do**
        Append camera detection to f_list.
        Retain 3D position and velocity estimate from LiDAR detection.
    end for
    for unmatched detections **do**
        Append to f_list.
    end for
    Update SORT tracker with f_list.
    Return states, class IDs, and track IDs.
end while

For applications that require 3D position and velocity, only fused, or LiDAR-only detections contain 3D information. Thus, camera-only detections would be excluded from f_list.

7.4 Evaluation

The camera-LiDAR fusion pipeline results are processed as mentioned in Section 6.4. Table 7.1 shows the HOTA score, sub-metric accuracies, and number of detections.

Table 7.1: Tracking performance of fusion pipeline on the CMHT Traffic dataset.

Sequence	**HOTA ↑**	**DetA ↑**	**AssA ↑**	**LocA ↑**	**Dets**	**GT Dets**	**IDs**	**GT IDs**
May10_R5	67.6	69.6	66.2	84.1	352	370	19	12
May10_R7	**76.7**	76.2	77.9	86.8	190	192	11	8
Dec7_R1	66.3	67.8	65.2	84.7	1891	2071	55	39
Dec14_R1	56.1	49.0	66.5	80.6	522	663	25	23
Oct18_R1	**39.1**	37.3	41.0	77.0	593	633	39	20
Oct18_R9	42.0	33.4	53.7	72.4	237	348	7	6
Combined	59.8	57.0	63.6	82.6	3785	4277	156	108

Figure 7.3a shows an example rooftop image with detections and track IDs. The orange label is a LiDAR-only result, the green labels are camera-only detections, and the blue labels represent objects detected by LiDAR and camera. The number at the bottom right of every bounding box represents the track ID. Figure 7.3b shows a nighttime roadside sample result. In both examples, the sensors support each other when the other fails to detect an object. This is apparent in Figure 7.3a with the occluded white vehicle and Figure 7.3b with the pedestrian on the opposite sidewalk.

(a) Rooftop daytime fusion result.

(b) Roadside nighttime fusion result.

(c) Roadside nighttime fusion result.

(d) Rooftop evening fusion result.

Figure 7.3: Qualitative Camera-LiDAR Fusion results. The green, blue, and orange label represent camera only, LiDAR only, and fused results respectively.

7.5 Summary

This chapter discusses camera-LiDAR fusion. First, a review of sensor fusion and relevant works is provided. Then the proposed camera-LiDAR fusion framework is presented, it leverages the camera's classification ability and the LiDAR's low-light performance, achieving a HOTA score of 59.8 on the CMHT Traffic dataset. The fusion pipeline will be compared with the single sensor methods in Chapter 9.

Chapter 8

System Design and Implementation

8.1 Introduction

This section details the design and implementation of the proposed camera-LiDAR fusion system. The system is designed with a modular architecture, allowing easy integration and algorithm evaluation, while maintaining a fast runtime that is critical for real-world deployment. The modular design not only simplifies development and testing of new components but also ensures that the system can be easily adapted to different algorithms and hardware configurations.

Figure 8.1 shows the diagram of the combined camera-LiDAR fusion system. On the left, the top and bottom blocks represent the camera and LiDAR object detection pipelines. The right-most block is the fusion module, which outputs the final track list.

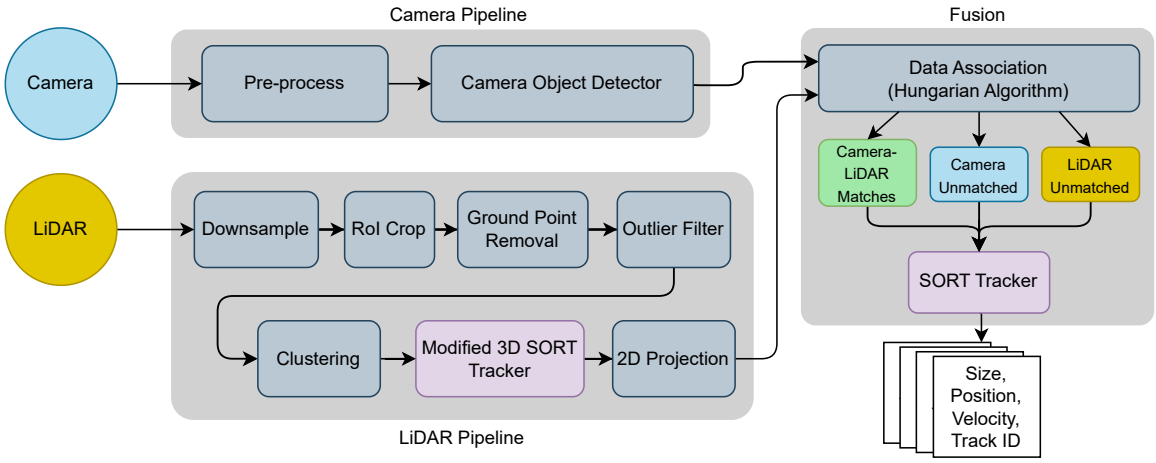


Figure 8.1: Diagram of the proposed Camera-LiDAR Fusion pipeline.

Part of the objective is to create a system that performs online processing at 10 Hz on minimal hardware. The host computer used for development and testing has a modest GPU, but still has greater computing resources compared to many embedded edge devices. Table 8.1 outlines the specifications of the host computer used to run the system. The GPU has 4 GB of VRAM, which is modest compared to the GPUs used in current 2D and 3D perception work that often have 12–32 GB.

Table 8.1: Host computer specifications.

Part	**Specification**
CPU	i7-11800H @ 2.30GHz × 16
RAM	32 GB
GPU	NVIDIA T1200 4 GB

The goal of the system to minimize the runtime, which enables computationally weaker systems to achieve the minimum 10 Hz processing rate. Chapter 9 will present and discuss the timing results of the proposed framework.

8.2 Software

The Robot Operating System 2 (ROS 2) is used to manage the parallel processes in the pipeline and provide a middleware for inter-process communication. The system consists of modules written in **C++** and **Python**, and libraries such as the Point Cloud Library (PCL) [11], Eigen [109], NumPy [110], PyTorch, and others.

Since ROS 2 supports nodes implemented in Python and C++, selecting a programming language was based on development effort and performance. The LiDAR object detection node was written in C++, primarily since the PCL is native to C++ and there is high quality documentation and examples. Furthermore, it would be beneficial to have a compiled program executing the code instead of Python for computationally heavy modules. However, for computer vision tasks, many existing modules and libraries exist in Python, thus to save time and effort the decision was made to write them in Python. Moreover, many Python function calls are wrappers for C++, depending on the task the marginal increase in speed is outweighed by the extra development effort.

The ROS 2 framework has many features that assist project development. Firstly, it is an open-source project with a large community, which means that various bugs and issues may have existing solutions. And many tutorials and documents can be easily accessed online. Secondly, ROS 2 comes with many useful programs to help develop complex systems such as Rviz2, rosbag, and RQT. Rviz2 is a graphical visualization tools with built-in support for 3D point clouds, images, and other forms of data, this is useful for debugging and demonstrating the algorithms, as seen in many of the figures of this thesis. Rosbag is a tool that allows for recording and playback of data, it publishes the recorded data similarly to the original data source.

This allows a developer to use recorded data or a live sensor without changing source code. Furthermore, rosbag has function to slow or speedup the data playback rate, this assists in debugging and evaluating a system. RQT is a collection of graphical tools that help understand the system, one example is the graph tool which provides a system graph, with processes as the nodes, and the data as the edges. Finally, ROS 2 provides inter-process communication and synchronization of independent processes and threads. All nodes are automatically given their own process, which reduces development effort compared to a custom solution.

NVIDIA TensorRT was used to speed up YOLOv5 inference times. This works by converting the PyTorch model weights into an *engine* file and using specialized software that is optimized for NVIDIA GPUs. TensorRT increased the inference rate by two times compared to the default PyTorch implementation.

8.2.1 Design

The design of this system followed software engineering principles. Modularity and separation of concerns were primary influences, since they enable ease of algorithm swapping and testing. Figure 8.2 shows the module interfaces for each node in the system, each implements a single high-level task such as detection, tracking or classification.

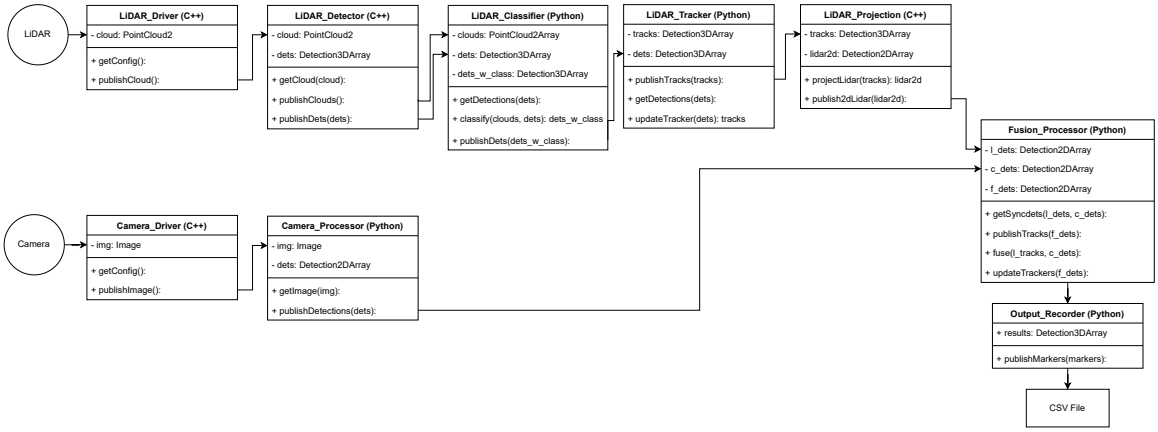


Figure 8.2: Module interfaces of for the camera-LiDAR fusion framework.

This project leverages ROS 2 to increase reusability, adaptability and understandability for future research and development. ROS 2 provides inter process communication via topics. Any node that publishes or subscribes to a specific topic can share or consume the information. Moreover, the topic names and other variables can be parameterized, allowing a node to be instantiated with different parameters at runtime. Multiple system configurations can be easily created to evaluate different detectors, trackers, and fusion strategies. Figure 8.3 shows a graph of the Camera-LiDAR fusion pipeline created by the ROS 2 *RQT Graph* tool, the edges of the graph are labelled with the message that is passed between nodes.

Another important part of the system is logging data and evaluating performance regardless of its configuration. With ROS 2 it's trivial to create another node that subscribes messages containing intermediate or final results and store them for offline evaluation.

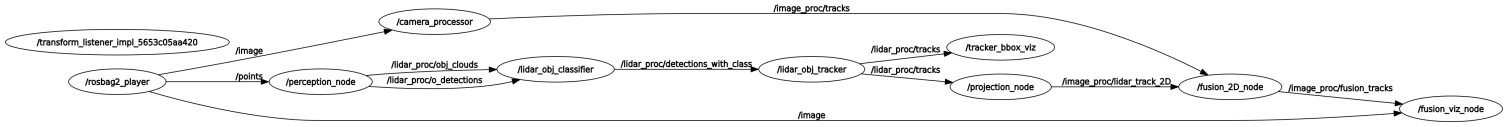


Figure 8.3: Diagram of the ROS 2 system graph for the fusion pipeline.

As shown in the above diagrams, the camera and LiDAR processing occurs in parallel. The sensors are pseudo synchronized, the average temporal misalignment is between 5-10 ms. Thus, the pipelines start at the same time, but have different processing times. The fusion node, which consumes the processed sensor detections, waits for both sets of detections before fusing and tracking the data. The original timestamps from each sensor are propagated through the processing steps so that the fusion node does not erroneously fuse data from a previous time step. If 90 ms passes before a synchronized set of data arrives, the frame is dropped.

8.3 Tools

Version control is critical in a large project to provide backups and opportunities to experiment on other branches without disturbing the main code. Version control is managed via Git and GitHub. Git provides the ability to revert files to previous states and provides tools to manage changes to software. GitHub provides online hosting and graphical access to the repository.

Ground Truth Labelling. The collected image data was labelled to create a 2D ground truth to evaluate the camera, LiDAR, and Camera-LiDAR pipelines. *Supervisely* [111] is an online platform that was selected for this labelling task. Since the ground truth required 2D bounding boxes and track IDs, a video is created with consecutive images, and the system provides tools for drawing bounding boxes and

track maintenance.

Metric Calculation. Many popular metrics have public repositories to compute metrics using custom data. HOTA calculations were made using the publicly available repository [112]. mAP was computed using [113], custom Python scripts converted the ground truth and detection results into the required COCO format. Classification metrics and the confusion matrix were calculated using Sciki-Learn [114].

8.4 Summary

This chapter provides an overview of the tools and implementation of the entire camera-LiDAR fusion system. ROS 2 is used to provide multiprocessing and inter-process communication for components of the system. This enables future algorithms and strategies to be easily integrated and evaluated. By effectively using a version control system, following best practices and creating documentation, future re-use and modification is simplified.

Chapter 9

System Evaluation

This chapter provides a comprehensive evaluation of the proposed camera-LiDAR fusion framework. The assessment is designed to validate the system's performance against real-world data, evaluating its robustness, accuracy, and speed. First, the proposed system will be compared against camera-only and LiDAR-only pipelines using the CMHT Traffic dataset and HOTA. Then the runtime results and computational resources are presented to determine if the real-time requirement is met.

9.1 Pipeline Comparison

Table 9.1 shows the aggregated HOTA scores for the LiDAR-only, camera-only, and camera-LiDAR fusion pipelines, as shown in Sections 6.4 and 7.4. Camera-LiDAR outperforms the other methods by a margin of 3%. This result is expected since detections from both modalities are combined and tracked.

Table 9.1: Summary of pipeline tracking performance on the CMHT Traffic dataset.

Sequence	**HOTA ↑**	**DetA ↑**	**AssA ↑**	**LocA ↑**	**Dets**	**GT Dets**	**IDs**	**GT IDs**
LiDAR Only	31.9	27.6	38.3	65.9	2959	4277	**130**	108
Camera Only	56.8	55.6	58.9	**84.5**	3305	4277	153	108
Camera-LiDAR	**59.8**	**57.0**	**63.6**	82.6	**3785**	4277	156	108

Figure 9.1 plots Detection Accuracy (DetA) vs. Assignment Accuracy (AssA) for each pipeline, the contour lines denote constant HOTA score. The camera and fusion results are similar in detection accuracy (1.4 %), however the fusion has significantly association accuracy (4.7 %). This suggests that the camera-LiDAR fusion is able to reduce identity switching and track fragmentation. One reason for this is that the LiDAR detections supplement detections that the camera misses, which allows the track to continue without fragmentation.

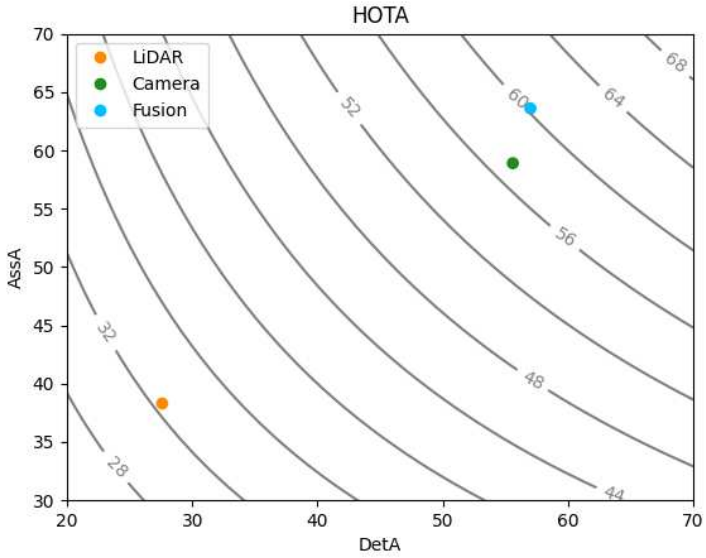
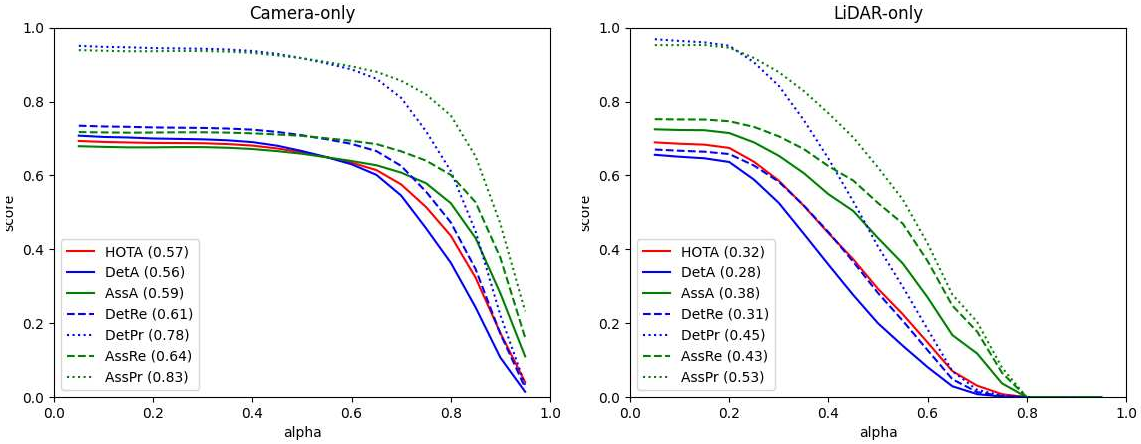


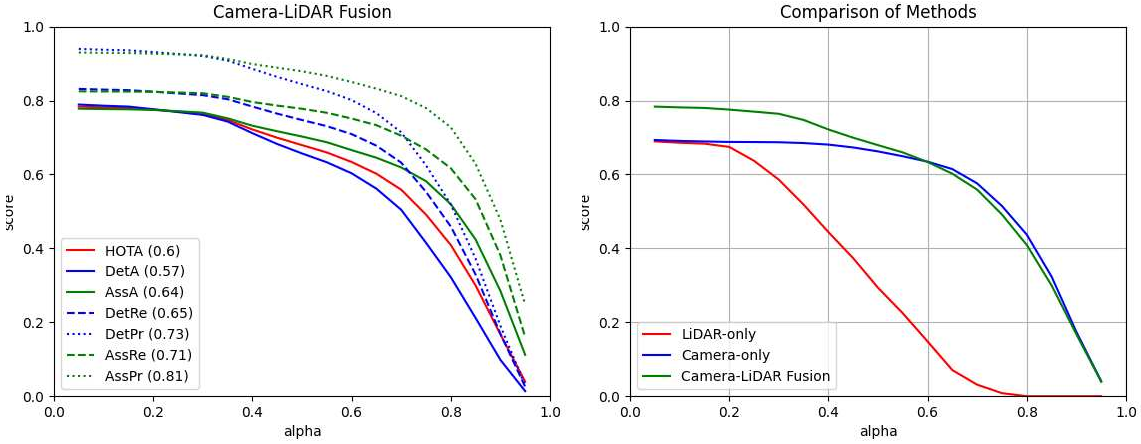
Figure 9.1: Comparing tracking pipelines. Grey curves are contours of constant score.

When comparing multi-object detection and tracking results, it is important to consider the effect of the localization threshold, alpha. Since HOTA is computed using a threshold range from [0.05, 0.95], systems with a constant localization error will significantly suffer in the upper range. Moreover, in the traffic monitoring context, a bounding box result with an IoU score of 0.95 is not necessary. Figure 9.2 plots the HOTA score for each alpha value. The alpha value controls the localization threshold that determines if two detections are a match, i.e., the IoU score between a detection and the ground truth exceeds alpha. All trackers will have a lower score as alpha is increased, since the stricter threshold reduces the number of valid detections used to calculate the score. Figure 9.2b shows that the LiDAR-only results are comparable to the camera and fusion methods for alpha values up until 0.2. This is expected since the LiDAR method loses localization accuracy from the 3D to 2D projection, calibration errors, and temporal misalignment. Therefore, the LiDAR-only performance is understated with the default HOTA score. It is also interesting to note that the fusion method only exceeds the camera-only in the [0.0, 0.5] range.



(a) Camera-based pipeline using YOLOv5m + SORT.

(b) LiDAR-based pipeline using the proposed unsupervised method + modified SORT.



(c) Camera-LiDAR fusion-based pipeline using YOLOv5m and the proposed unsupervised LiDAR method + SORT.

(d) HOTA at each alpha level for all methods.

Figure 9.2: Plots of HOTA score vs alpha. A lower alpha value represents a lower localization threshold.

9.2 Runtime Results

Since traffic monitoring is an online task, runtime is an important system performance measure. Table 9.2 details the time in milliseconds (ms) and the implementation language for each component of the LiDAR pipeline. All tests were performed on the hardware mentioned detailed in Section 8.1 and calculated using the mean from 200 frames from the Dec14_R1 sequence.

The tracking module is the slowest due to the rotated rectangle IoU calculation in the data association step. Implementing the tracking module in C++ would yield a significant speed-up. Overall, the LiDAR pipeline time is 12.9 ms per LiDAR frame. The runtime varies with the number of points, a larger RoI or denser point cloud will increase the runtime.

Table 9.2: LiDAR pipeline runtime results.

Module	**Runtime** (ms)	**Implementation**
Downsample	3.6	C++
Region of Interest Crop	1.8	C++
Ground Point Removal	0.5	C++
Outlier Filter	0.9	C++
Clustering	1.3	C++
Classification	0.9	Python3.8
2D Projection	0.2	C++
Tracking	3.7	Python3.8
Total	**12.9 (78 Hz)**	

Table 9.3 shows the runtime results and the processor that was used for the

camera-LiDAR fusion system components. Since the camera and LiDAR pipelines execute in parallel, the maximum of the two is added used. Overall, the system runtime is 21.8 ms (46 Hz). Although the LiDAR sensor publishes data at 10 Hz, it is important to minimize the runtime for multiple reasons. First, the remaining 78.2 ms can be used for higher level functions such as occupancy detection, counting, or near-miss detection. Second, the additional time can be used to implement more complex algorithms to increase accuracy. Finally, minimizing the runtime enables a wider range of hardware to run the system within the 100 ms requirement, including edge devices with integrated GPUs and slower or fewer processing cores.

Table 9.3: Fusion framework runtime. Total runtime is $\max(t_{camera}, t_{LiDAR}) + t_{fusion}$.

Module	Runtime (ms)	Processing Unit
LiDAR Pipeline	12.9	CPU (1 core @ 2.3 GHz)
Camera Pipeline	20.7	GPU (0.5 GB Used)
Fusion Module	1.1	CPU (1 core @ 2.3 GHz)
Total	**21.8 (46 Hz)**	

9.3 Summary

This chapter compared the overall MOT results between the sensor pipelines. The camera-LiDAR fusion system achieved the highest HOTA score. Then the runtime of the individual modules and overall system were presented. The overall runtime of the camera-LiDAR fusion pipeline is 21.8 ms (46 Hz), providing room for more complex algorithms or porting to a less powerful device.

Chapter 10

Conclusion and Future Work

10.1 Conclusion

Traffic monitoring systems will be important components of a safer future for all road-users. Reducing accidents, traffic, and pollution can be achieved with Vehicle-to-Everything communication and accurate road use statistics. To accomplish this, Multi Object Tracking (MOT) systems need to be robust to lighting and environmental conditions and operate at real-time on embedded systems. This thesis aimed to design and implement a lightweight camera-LiDAR fusion framework for static traffic monitoring applications. To achieve this, a custom LiDAR pipeline including object detection, classification, and tracking was presented. The fusion module combined detections from a pre-trained CNN with the LiDAR pipeline to produce a robust 2D tracking system. The camera-LiDAR fusion system achieved the highest HOTA score against camera-only and LiDAR-only MOT pipelines on a custom dataset containing varying lighting conditions. The pipeline achieved an overall throughput of 46 Hz with an entry-level GPU.

The following primary objectives were achieved:

- Design of a lightweight, end-to-end Camera-LiDAR fusion system for traffic monitoring, that performs multi-object detection and tracking.
- Implementation of the above system that achieves real-time processing speed on live and recorded data. An interface simplifies algorithm substitution and evaluation.
- Evaluation of the above system comparing camera, LiDAR, and camera-LiDAR fusion approaches. This was accomplished via assembly of a test fixture for capturing a synchronized dataset of LiDAR point clouds and images in multiple lighting scenarios. Hand-labelled detections and track IDs were created for the images.

10.2 Future Work

The work throughout this thesis has yielded a simple yet effective baseline for LiDAR-based detection and camera-LiDAR fusion. Since the algorithms are built upon a modular software system, it is easy to extend this software and compare it to new and existing datasets. Future work can be categorized as extensions and upgrades of the algorithms themselves, and the integration of this system in V2X or ITS applications.

10.2.1 Camera Object Detection

2D Detection

Since many modern tracking algorithms are part of the detection-based tracking paradigm, a logical extension would be to increase the detection performance of both the camera and LiDAR pipelines. For the camera, performing transfer learning or fine-tuning the model weights to a traffic monitoring dataset will increase detection accuracy for road users. The default YOLOv5 model weights trained on the COCO dataset generalize well, however, the size and angle of pedestrians and cars combined with varying weather conditions challenge the model. Another focus can include increasing the inference speed of the model via pruning or quantization. These methods reduce the number of computations and hence time required to process an image.

3D Detection

3D Camera detections enable 3D fusion between camera and LiDAR detections. 3D Detections are significantly more valuable for V2X applications, since AVs use 3D perception and path planning pipelines.

10.2.2 LiDAR Object Detection

For the LiDAR pipeline, a deep learning-based 3D object detection model could be trained on a road-side dataset to increase the accuracy of the detections. Furthermore, the integration of multiple LiDAR sensors from different perspectives would allow a system to track objects over a larger FoV, and increases the point cloud density, increasing the quality of detection and track results. Finally, exploring other LiDAR

sensors, such as solid-state LiDAR with higher resolution and FMCW LiDAR that include point velocity, could improve detection performance.

10.2.3 Data Fusion

This thesis presents a simple fusion implementation, further work could include developing a more complex fusion framework that includes confidence scores and probability of detections. Moreover, experimenting with data-level fusion by combining camera and LiDAR information with a DNN could improve detection accuracy by reducing information loss.

10.2.4 Dataset Diversity

Since a traffic monitoring system is expected to function consistently at all times, a diverse dataset should contain most environmental conditions expected during standard use. Rain, snow, and fog, during both day and night conditions for evaluation would provide a better understanding of system performance. Furthermore, labelled data from these diverse conditions can be used to fine tune models, as mentioned above.

10.2.5 ITS and V2X Integration

Other future work includes integrating the object tracking results into an ITS system. Possible experiments include integration with street light controllers to dynamically update pedestrian crossing times based on intersection occupancy, or integration with a vehicle's on-board detection system to create a Vehicle-to-Infrastructure system.

Thank you for reading this far.

Bibliography

[1] Ilina Chipilska. What Is An Intelligent Transport System And How Does It Work?, September 2023.

[2] Ezekiel Bokolonga, Martin Hauhana, Nicholas Rollings, David Aitchison, Mansour H. Assaf, Sunil R. Das, Satyendra N. Biswas, Voicu Groza, and Emil M. Petriu. A compact multispectral image capture unit for deployment on drones. In *2016 IEEE International Instrumentation and Measurement Technology Conference Proceedings*, pages 1–5, May 2016.

[3] OpenCV. *The OpenCV Reference Manual*, 2.4.13.7 edition, April 2014.

[4] Zhen Xue, Liangliang Zhang, and Bo Zhai. Multiscale Object Detection Method for Track Construction Safety Based on Improved YOLOv5. *Mathematical Problems in Engineering*, 2022:1–10, August 2022.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017.

[6] Max Ferguson, Ronay Ak, Yung-Tsun Tina Lee, and Kincho H. Law. Automatic localization of casting defects with convolutional neural networks. In *2017 IEEE*

International Conference on Big Data (Big Data), pages 1726–1735, December 2017.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[8] Xuan-Thuy Vo and Kang-Hyun Jo. A review on anchor assignment and sampling heuristics in deep learning-based object detection. *Neurocomputing*, 506:96–116, September 2022.

[9] Xin Nie, Meifang Yang, and Ryan Wen Liu. Deep neural network-based robust ship detection under different weather conditions. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 47–52, 2019.

[10] Juan Terven and Diana Cordova-Esparza. A Comprehensive Review of YOLO: From YOLOv1 and Beyond. In *Under review in ACM Computing Surveys*, October 2023.

[11] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4, May 2011. ISSN: 1050-4729.

[12] Sergio González, Salvador García, Javier Del Ser, Lior Rokach, and Francisco Herrera. A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities. *Information Fusion*, 64:205–237, December 2020.

[13] Deandra Aulia and Hendri Murfi. XGBoost in handling missing values for life insurance risk prediction. *SN Applied Sciences*, 2, August 2020.

[14] Sensor Data — Ouster Sensor Docs documentation.

[15] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, imyhxy, Lorna, Zeng Yifu), Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation, November 2022.

[16] Eiman Nuaimi, Hind Neyadi, Nader Mohamed, and Jameela Al-Jaroodi. Applications of big data to smart cities. *Journal of Internet Services and Applications*, 6, August 2015.

[17] Margarita Martínez-Díaz and Francesc Soriguera. Autonomous vehicles: theoretical and practical challenges. *Transportation Research Procedia*, 33:275–282, January 2018.

[18] Walter Zimmer, Jialong Wu, Xingcheng Zhou, and Alois C. Knoll. Real-Time And Robust 3D Object Detection with Roadside LiDARs. In Constantinos Antoniou, Fritz Busch, Andreas Rau, and Mahesh Hariharan, editors, *Proceedings of the 12th International Scientific Conference on Mobility and Transport*, Lecture Notes in Mobility, pages 199–219, Singapore, 2023. Springer Nature.

[19] Csaba Benedek, Andras Majdik, Balazs Nagy, Zoltan Rozsa, and Tamas Sziranyi. Positioning and perception in LIDAR point clouds. *Digital Signal Processing*, 119:103193, December 2021.

[20] Stefan Muckenhuber, Hannes Holzer, and Zrinka Bockaj. Automotive Lidar Modelling Approach Based on Material Properties and Lidar Capabilities. *Sensors (Basel)*, 20(11):3309, June 2020.

[21] Diego Pierrottet, Farzin Amzajerdian, Larry Petway, Bruce Barnes, George Lockard, and Manuel Rubio. Linear FMCW Laser Radar for Precision Range and Vector Velocity Measurements. *MRS Proceedings*, 1076, January 2008.

[22] Taohua Zhou, Mengmeng Yang, Kun Jiang, Henry Wong, and Diange Yang. MMW Radar-Based Technologies in Autonomous Driving: A Review. *Sensors*, 20:7283, December 2020.

[23] G.M. Brooker, S. Scheding, M.V. Bishop, and R.C. Hennessy. Development and application of millimeter wave radar sensors for underground mining. *IEEE Sensors Journal*, 5(6):1270–1280, December 2005. Conference Name: IEEE Sensors Journal.

[24] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A Multimodal Dataset for Autonomous Driving. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11618–11628, Seattle, WA, USA, June 2020. IEEE.

[25] Whye Kit Fong, Rohit Mohan, Juana Valeria Hurtado, Lubing Zhou, Holger

Caesar, Oscar Beijbom, and Abhinav Valada. Panoptic nuscenes: A large-scale benchmark for lidar panoptic segmentation and tracking. *IEEE Robotics and Automation Letters*, 7(2):3795–3802, 2022.

[26] Andreas Geiger, P Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: the KITTI dataset. *The International Journal of Robotics Research*, 32:1231–1237, September 2013.

[27] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurélien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2443–2451, 2020.

[28] Pengchuan Xiao, Zhenlei Shao, Steven Hao, Zishuo Zhang, Xiaolin Chai, Judy Jiao, Zesong Li, Jian Wu, Kai Sun, Kun Jiang, Yunlong Wang, and Diange Yang. Pandaset: Advanced sensor suite dataset for autonomous driving. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 3095–3101, 2021.

[29] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, Tiffany Fernandez, Martin Jänicke, Sudesh Mirashi,

Chiragkumar Savani, Martin Sturm, Oleksandr Vorobiov, Martin Oelker, Sebastian Garreis, and Peter Schuberth. A2D2: Audi Autonomous Driving Dataset, April 2020. arXiv:2004.06320 [cs, eess].

[30] Christian Creß, Walter Zimmer, Leah Strand, Maximilian Fortkord, Siyi Dai, Venkatnarayanan Lakshminarasimhan, and Alois Knoll. A9-Dataset: Multi-Sensor Infrastructure-Based Dataset for Mobility Research. In *2022 IEEE Intelligent Vehicles Symposium (IV)*, pages 965–970, June 2022.

[31] Walter Zimmer, Christian Creß, Huu Nguyen, and Alois Knoll. *A9 Intersection Dataset: All You Need for Urban 3D Camera-LiDAR Roadside Perception*. June 2023.

[32] Steffen Busch, Christian Koetsier, Jeldrik Axmann, and Claus Brenner. LUMPI: The Leibniz University Multi-Perspective Intersection Dataset. In *2022 IEEE Intelligent Vehicles Symposium (IV)*, pages 1127–1134, June 2022.

[33] Haibao Yu, Yizhen Luo, Mao Shu, Yiyi Huo, Zebang Yang, Yifeng Shi, Zhenglong Guo, Hanyu Li, Xing Hu, Jirui Yuan, and Zaiqing Nie. Dair-v2x: A large-scale dataset for vehicle-infrastructure cooperative 3d object detection. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21329–21338, 2022.

[34] Huanan Wang, Xinyu Zhang, Zhiwei Li, Jun Li, Kun Wang, Zhu Lei, and Ren Haibing. IPS300+: a Challenging multi-modal data sets for Intersection Perception System. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2539–2545, Philadelphia, PA, USA, May 2022. IEEE Press.

[35] Deng Yongqiang, Wang Dengjiang, Cao Gang, Ma Bing, Guan Xijia, Wang Yajun, Liu Jianchao, Fang Yanming, and Li Juanjuan. *BAAI-VANJEE Roadside Dataset: Towards the Connected Automated Vehicle Highway technologies in Challenging Environments of China*. May 2021.

[36] Md Zahangir Alom, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C. Van Esesn, Abdul A. S. Awwal, and Vijayan K. Asari. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches, September 2018. arXiv:1803.01164 [cs].

[37] Ibrahem Kandel and Mauro Castelli. Transfer Learning with Convolutional Neural Networks for Diabetic Retinopathy Image Classification. A Review. *Applied Sciences*, 10:2021, March 2020.

[38] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[39] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

[40] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, January 1985.

[41] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva

Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, Lecture Notes in Computer Science, pages 740–755, Cham, 2014. Springer International Publishing.

[42] Thorsten Hoeser and Claudia Kuenzer. Object Detection and Image Segmentation with Deep Learning on Earth Observation Data: A Review-Part I: Evolution and Recent Trends. *Remote Sensing*, 12, May 2020.

[43] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521:436–44, May 2015.

[44] Shuying Liu and Weihong Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.

[45] Xavier Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256, January 2010.

[46] Zihong Shi. Object Detection Models and Research Directions. In *2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, pages 546–550, January 2021.

[47] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[48] Ross Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[49] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[50] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective Search for Object Recognition. *International Journal of Computer Vision*, 104, 2013.

[51] O. Hmidani and E. M. Ismaili Alaoui. A comprehensive survey of the R-CNN family for object detection. In *2022 5th International Conference on Advanced Communication Technologies and Networking (CommNet)*, pages 1–6, December 2022. ISSN: 2771-7402.

[52] Walter Zimmer, Emec Ercelik, Xingcheng Zhou, Xavier Jair Diaz Ortiz, and Alois Knoll. A Survey of Robust 3D Object Detection Methods in Point Clouds, March 2022. arXiv:2204.00106 [cs].

[53] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, Las Vegas, NV, USA, June 2016. IEEE.

[54] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger, December 2016.

[55] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement, April 2018.

[56] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection, April 2020.

[57] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1571–1580, 2020.

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, Lecture Notes in Computer Science, pages 346–361, Cham, 2014. Springer International Publishing.

[59] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018.

[60] Ho Lee and Benjamin Coifman. Side-Fire Lidar-Based Vehicle Classification. *Transportation Research Record: Journal of the Transportation Research Board*, 2308(1):173–183, January 2012.

[61] Tianya Zhang and Peter J. Jin. Roadside LiDAR Vehicle Detection and Tracking Using Range and Intensity Background Subtraction. *Journal of Advanced Transportation*, 2022:e2771085, April 2022. Publisher: Hindawi.

[62] Jianqing Wu, Chen Lv, Rendong Pi, Zhaoyou Ma, Han Zhang, Renjuan Sun, Yanjie Song, and Kai Wang. A Variable Dimension-Based Method for Roadside LiDAR Background Filtering. *IEEE Sensors Journal*, 22(1):832–841, January 2022. Conference Name: IEEE Sensors Journal.

[63] John Shackleton, Brian VanVoorst, and Joel Hesch. Tracking People with a 360-Degree Lidar. In *2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 420–426, August 2010.

[64] Jianqing Wu, Hao Xu, and Jianying Zheng. Automatic background filtering and lane identification with roadside LiDAR data. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, October 2017. ISSN: 2153-0017.

[65] Walter Zimmer, Joseph Birkner, Marcel Brucker, Huu Tung Nguyen, Stefan Petrovski, Bohan Wang, and Alois C. Knoll. InfraDet3D: Multi-Modal 3D Object Detection based on Roadside Infrastructure Camera and LiDAR Sensors, April 2023. arXiv:2305.00314 [cs].

[66] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[67] Radu Bogdan Rusu. Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments. *Künstl Intell*, 24(4):345–348, November 2010.

[68] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise.

In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[69] Bo Li. 3D fully convolutional network for vehicle detection in point cloud. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1513–1518, September 2017. ISSN: 2153-0866.

[70] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3Deep: Fast Object Detection in 3D Point Clouds Using Efficient Convolutional Neural Networks, March 2017. arXiv:1609.06666 [cs].

[71] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. Complex-YOLO: Real-time 3D Object Detection on Point Clouds, September 2018. arXiv:1803.06199 [cs].

[72] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018.

[73] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. PointPillars: Fast Encoders for Object Detection from Point Clouds, May 2019. arXiv:1812.05784 [cs, stat].

[74] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[75] Zetong Yang, Yanan Sun, Shu Liu, and Jiaya Jia. 3DSSD: Point-Based 3D Single Stage Object Detector. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11037–11045, June 2020. ISSN: 2575-7075.

[76] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–779, 2019.

[77] Yusheng Xu, Xiaohua Tong, and Uwe Stilla. Voxel-based representation of 3D point clouds: Methods, applications, and its potential use in the construction industry. *Automation in Construction*, 126:103675, June 2021.

[78] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[79] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. *SIGMOD Rec.*, 28(2):49–60, jun 1999.

[80] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[81] Evelyn Fix and Joseph L. Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties, 1951.

[82] J.R. Quinlan. Decision trees and decision-making. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):339–346, March 1990. Conference Name: IEEE Transactions on Systems, Man, and Cybernetics.

[83] Corrado Gini. Variabilità e mutabilità. contributo allo studio delle distribuzioni e delle relazioni statistiche. *Bologna: C. Cuppini*, 1912.

[84] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[85] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

[86] Zhenyu Lin, Masafumi Hashimoto, Kenta Takigawa, and Kazuhiko Takahashi. Vehicle and Pedestrian Recognition Using Multilayer Lidar based on Support Vector Machine. In *2018 25th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pages 1–6, November 2018.

[87] Alex Teichman, Jesse Levinson, and Sebastian Thrun. Towards 3D object recognition via classification of arbitrary object tracks. In *2011 IEEE International Conference on Robotics and Automation*, pages 4034–4041, May 2011. ISSN: 1050-4729.

[88] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893, San Diego, CA, USA, 2005. IEEE.

[89] David Hand and Peter Christen. A note on using the F-measure for evaluating record linkage algorithms. *Statistics and Computing*, 28(3):539–547, May 2018.

[90] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, and Tae-Kyun Kim. Multiple Object Tracking: A Literature Review. *Artificial Intelligence*, 293:103448, April 2021. arXiv:1409.7618 [cs].

[91] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, March 1960.

[92] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800020109.

[93] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, December 1987.

[94] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple Online and Realtime Tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468, September 2016. arXiv:1602.00763 [cs].

[95] Jinkun Cao, Jiangmiao Pang, Xinshuo Weng, Rawal Khirodkar, and Kris Kitani. Observation-Centric SORT: Rethinking SORT for Robust Multi-Object Tracking. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9686–9696, Vancouver, BC, Canada, June 2023. IEEE.

[96] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649, September 2017. ISSN: 2381-8549.

[97] Yunhao Du, Zhicheng Zhao, Yang Song, Yanyun Zhao, Fei Su, Tao Gong, and Hongying Meng. StrongSORT: Make DeepSORT Great Again. *IEEE Transactions on Multimedia*, 25:8725–8737, 2023. Conference Name: IEEE Transactions on Multimedia.

[98] Keni Bernardin and Rainer Stiefelhagen. Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics. *EURASIP Journal on Image and Video Processing*, 2008(1):1–10, December 2008. Number: 1 Publisher: SpringerOpen.

[99] Jonathon Luiten, Aljosa Osep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. HOTA: A Higher Order Metric for Evaluating Multi-object Tracking. *International Journal of Computer Vision*, 129(2):548–578, February 2021.

[100] Luis Gressenbuch, Klemens Esterle, Tobias Kessler, and Matthias Althoff. MONA: The Munich Motion Dataset of Natural Driving. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2093–2100, October 2022.

[101] Qin Tang, Jing Liang, and Fangqi Zhu. A comparative review on multi-modal sensors fusion based on deep learning. *Signal Processing*, 213:109165, December 2023.

[102] Zhijian Liu, Haotian Tang, Alexander Amini, Xinyu Yang, Huizi Mao, Daniela Rus, and Song Han. BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation, June 2022. arXiv:2205.13542 [cs].

[103] Yingwei Li, Adams Wei Yu, Tianjian Meng, Ben Caine, Jiquan Ngiam, Daiyi Peng, Junyang Shen, Bo Wu, Yifeng Lu, Denny Zhou, Quoc V. Le, Alan Yuille, and Mingxing Tan. DeepFusion: Lidar-Camera Deep Fusion for Multi-Modal 3D Object Detection, March 2022. arXiv:2203.08195 [cs].

[104] Haibin Liu, Chao Wu, and Huanjie Wang. Real time object detection using LiDAR and camera fusion for autonomous driving. *Scientific Reports*, 13(1):8056, May 2023.

[105] Numan Senel, Klaus Kefferpütz, Kristina Doycheva, and Gordon Elger. Multi-sensor data fusion for real-time multi-object tracking. *Processes*, 11(2), 2023.

[106] Huilin Yin, Yu Lu, Jia Lin, Markus Schratter, and Daniel Watzenig. Multi-object tracking with object candidate fusion for camera and lidar data. In *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2965–2970, 2023.

[107] Sanketh Ailneni, Sudesh Kumar Kashyap, VPS Naidu, Amitabh Saraf, and Nandan K Sinha. Airborne Multi Target Track to Track Fusion of Radar and IRST for Advanced Multi Role Combat Aircrafts. *IFAC-PapersOnLine*, 55(22):224–229, January 2022.

[108] Harrison Taylor, Liam Hiley, Jack Furby, Alun Preece, and Dave Braines.

VADR: Discriminative Multimodal Explanations for Situational Understanding. In *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*, pages 1–8, July 2020.

[109] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[110] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[111] Supervisely. Unified OS/Platform for Computer Vision.

[112] Arne Hoffhues Jonathon Luiten. Trackeval. https://github.com/JonathonLuiten/TrackEval, 2020.

[113] COCO. cocoapi. https://github.com/cocodataset/cocoapi/tree/master, 2020.

[114] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.