5-9-2024

Autowaze: Automated, real-time, and accurate traffic analytics

Vijay Bajracharya
vb6487@rit.edu

Master's Thesis

Master of Science in Computer Science

Autowaze: Automated, real-time, and accurate traffic analytics

by

Vijay Bajracharya

Submitted to the
B. Thomas Golisano College of Computing and Information Sciences
Department of Computer Science
in partial fulfillment of the requirements for the
Master of Science Degree
at the Rochester Institute of Technology

Date: 5/9/2024

Abstract

A key factor leading to the inefficiency of transportation systems is the lack of real-time, fine-grained traffic analytics. An average motorist in the United States spends a significant amount of time looking for parking spots during their daily commute. Existing systems for traffic analytics are either not scalable to large metropolitan areas or require a *human-in-the-loop*. This project focuses on automating the detection of fine-grained traffic analytics. Leveraging on-board stereo cameras, Autowaze utilizes a crowd-sourced strategy to take time-windowed snapshots of the road containing 3D map points of the environment. These snapshots are used to extract changes in the environment which are uploaded to a central cloud server responsible for inferring traffic analytics such as vacant parking spots. Evaluations show that Autowaze correctly predicts the occupancy status of a parking spot 89% of the time. Moreover, the use of this system can lead to great reductions in search time for motorists looking for available parking as well as increased fuel efficiency and cost savings.

COMMITTEE LIST

Thesis Advisor and Committee Chair,

Fawad Ahmad

Committee Reader,

Zachary Butler

Committee Observer,

M. Mustafa Rafique

Acknowledgments

I would like to express my sincere gratitude to my faculty advisor, Dr. Fawad Ahmad, for mentoring me throughout the entire research process. Your insightful feedback and expertise in the field was instrumental to the success of this thesis. I would also like to thank members of my committee - Dr. Zachary Butler, and Dr. M. Mustafa Rafique, for their time and thoughtful feedback.

Contents

List of Figures

Chapter 1

Introduction

This chapter introduces the problem, presents the advantages and disadvantages of existing software, and outlines the significance of the thesis.

1.1 Overview

1.1.1 The Problem

The INRIX 2022 Global Traffic Scorecard analyzed traffic congestion in over a thousand cities world-wide and have reported a staggering 133 hours lost to congestion for the average driver in New York City [16]. As such, a large portion of the commute time is spent either stationary or moving well-below the posted speed limits. Avoiding roadwork and accidents, and finding street-level parking spaces are often major factors contributing to this congestion. In New York City alone, the average commuter spent 107 hours per year searching for vacant parking spots (Fig. 1.1) [15]. A primary cause for this is the lack of real-time, fine-grained, and accurate traffic analytics about on-road events. With access to such analytics, motorists and autonomous vehicles (AVs) of the future can make well-informed decisions about their routes. They can plan ahead of time using parking availability statistics instead of having to drive to the location before starting their search. This leads to better fuel efficiency, reduced travel times, and lower congestion rates in urban areas.

1.1.2 Widely Adopted Commercial Solutions

Waze is by far the most popular commercial system for traffic analytics among motorists. It provides drivers with real-time information to help them reach their destination efficiently. The app leverages data from its user base to detect

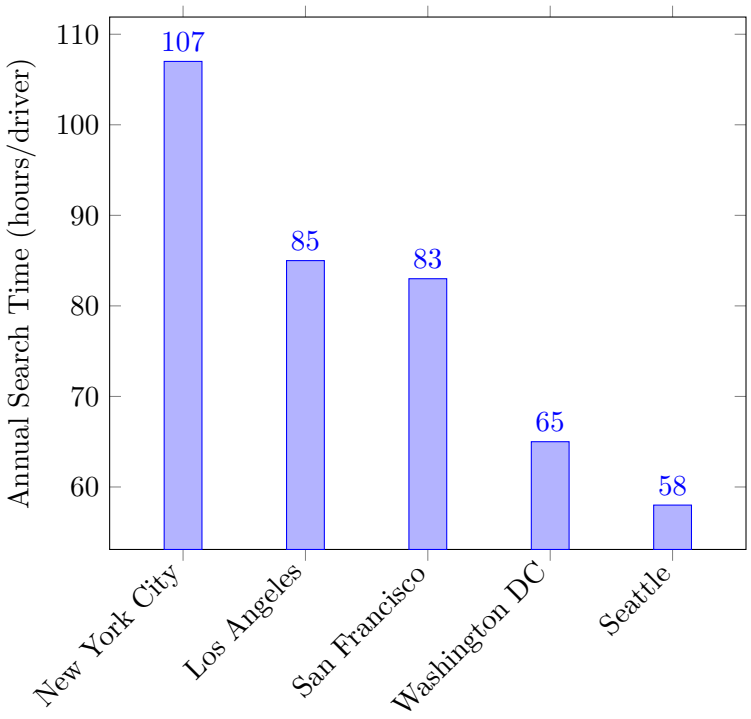


Figure 1.1: Hours spent searching for parking in major U.S. cities [15]

and report on-road events such as traffic jams, accidents, diversions, roadwork and speed traps [6].

At the core of Waze's functionality is its ability to gather location and speed data from users' smartphones as they drive. The app runs in the background and uses the positioning and gyroscopic sensors of a smartphone to determine vehicle status anonymously transmit information from thousands of users to a cloud server [6]. The cloud server analyzes this information to identify traffic patterns and recognize notable events such as congestion on the road. It then relays this information back to the users in the form of alerts or alternative routing suggestions.

In addition to the systems automatic service, it also encourages the user to manually report on-road events such as traffic accidents, road closures, and speed traps [6]. These crowd-sourced reports are verified using a voting system and shared to the rest of the user-base allowing everyone to make informed decisions.

The benefits of using Waze are numerous but it requires a *human-in-the-*

loop to fully realize on-road events and keep the user-base well-informed. Under New York State law, a driver cannot use hand-held mobile telephone or portable electronic devices [4]. Convictions for cell phone use while driving can not only result in fines but also points being added to a driver's DMV driving record [4]. Moreover, it cannot report fine-grained traffic events such as parking spot vacancy or occupied traffic spots due to imprecision in smartphone sensors. Any attempt to make reports on parking spot vacancy would have to account for both human errors and sensor errors.

1.1.3 Confined Solutions

Many parking facilities, particularly in crowded urban areas, have deployed sensor-based technology to detect parking spot availability. This type of parking spot detection systems are useful for confined spaces such as multi-level parking lots or garages at airports, shopping malls, or tourist attractions. These sensor-based approaches typically involve a network of small sensors installed at each individual parking spot or multiple overhead cameras that achieve a birds eye view of the parking lot. These sensors relay their findings to a central system which updates the parking spot vacancy statistics for the drivers to see. Magnetic sensors are often installed curbside to detect magnetic signatures of parked vehicles, allowing them to determine if a space is occupied or vacant [5]. Similarly, computer vision and deep learning models can also be used on camera footage to identify not only vacant parking spaces but also illegally parked vehicles [11].

While sensor-based approaches work well in confined locations, it faces significant challenges in terms of scalability and large scale practicality. The sheer number of parking spaces that would need to be equipped with sensors in a city-wide system would be prohibitively expensive. Additionally, these sensors need to be capable of withstanding the harsh conditions and maintenance costs can skyrocket within a few years of deployment.

1.1.4 Scalable Solutions In Literature

To address the challenge of city-wide scalability, recent research has explore the user of smartphone sensors as an alternative approach to determining key parking events for vehicles [10, 20, 23, 25]. For instance, the *ParkUs* system [10] collects data from the user's smartphone, including accelerometer, magnetometer, gyroscope, GPS speed, and GPS bearing readings. This sensor data is then processed through supervised machine learning classifiers to automatically determine if a vehicle is cruising or not cruising [10]. Using

this information, *ParkUs* determines the probabilistic parking availability of a given area visualized through a heat-map. Similarly, *Park Here!* [23] also uses accelerometer and gyroscope data from smartphones coupled with Bluetooth connectivity to identify transportation modes for the driver i.e. walking or driving. Parking-related events are then inferred by the system based on a sequence transportation mode transitions. *ParkSense* on the other hand leverages WiFi signature matching and rate of change of visible access points to determine if a driver is returning to their vehicle or driving away [20]. When a user pays for a parking spot, *ParkSense* captures their wireless signature on their smartphone. This signature is checked periodically to determine if a driver has returned to their vehicle.

While smartphone sensor-based approaches offer the scalability necessary for city-wide parking spot detection, they come with limitations such as sensor imprecision and the inability to determine parking occupancy at a spot-level.

1.2 Thesis Significance

This thesis seeks to automate the accurate detection and reporting of fine-grained traffic in real-time by setting the following objectives:

1. Eliminate the need for a *human-in-the-loop* in generating fine-grained traffic analytics;
2. Utilize pre-existing vehicle infrastructure to avoid extraneous costs of installing and maintaining additional edge infrastructure;
3. Develop an end-to-end prototype, from perception to information dissemination, capable of reporting parking spot vacancy statistics in real-time, and;
4. Address the system's ability to scale to city-wide adoption.

The core idea of the thesis is to find differences in time-windowed snapshots of the road and its surrounding to infer traffic analytics such as parking spot vacancy. The system defines two modes of operation: on-vehicle operations and cloud operations. On-vehicle operations occur at the vehicle nodes and it involves using pre-existing on-board cameras, compute resources, and wireless connectivity. As the vehicle drives through the street it captures snapshots of the environment and accumulates the information over pre-defined time-intervals. Snapshots are embodied as 3D maps of the environment containing distinctive features of the surrounding vehicles and structures. The vehicle

uses this 3D map to localize itself in the environment by matching its camera view to the 3D map. It does so by matching the features collected from the camera feed to the features present in the 3D map. In doing so, it can detect the addition of new features or absence of previously detected features from the environment which translates to finding differences between snapshots of the environment. These differences are uploaded to a centralized cloud service which updates its own version of the 3D map to infer traffic events based on a set of rules.

1.3 Core Research Challenge

The road and the surrounding is digitally represented using a 3D map. In order to reason about the high-level structure of objects in the environment, a dense 3D map with a large amount of features are required. However, performing operations within a dense 3D map is computationally expensive. Additionally, broadband mobile network speeds of today are not sufficient to support real-time inference. In contrast, sparse 3D maps only capture landmark features (e.g., the corner of a stop sign, cracks engraved in the road etc.) from the surrounding using feature extraction algorithms. Each feature is defined by its 3D position along with some feature signature.

The inherent sparsity of points in sparse 3D maps combined with the lack of embedded semantic information (e.g. the objects these features belong to) make it so that inferring high-level semantics from these maps using just their positions is severely limited. If we were to do a straightforward comparison to determine the differences between the 3D map and live sensor views, the result would be a collection of sparse 3D features. While precise in location, these features would provide little to no semantic information crucial for detecting high-level traffic events.

To this end, the key insight with this thesis is to utilize segmentation networks to enrich a sparse 3D map with semantic information such that high-level reasoning about traffic events is possible (Fig. 1.2).

1.4 Thesis Contributions

This thesis will make the following contributions:

1. Enhance the resourcefulness of sparse feature-based 3D map by augmenting each feature with contextual clues

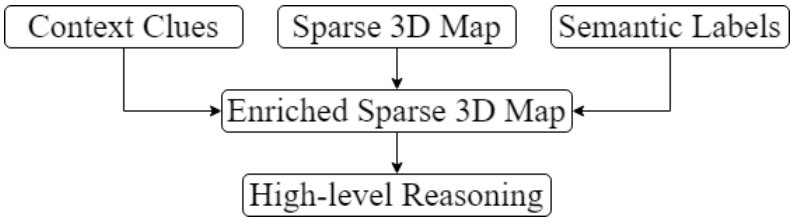


Figure 1.2: Enhancement of sparse 3D maps

2. Infer fine-grained traffic analytics with sparse 3D maps by means of a robust *semantic map diff* operation at the map-point level

3. Improve robustness to external factors such as occlusions and lightning conditions by using time-windowed snapshots of the environment

4. Present a novel approach for segmenting the point cloud based on contextual clues rather than statistical clustering

Chapter 2

Background

2.1 3D Map

A 3D map is a digital representation of a physical environment that captures depth, height, and width of terrain and the objects within it. Unlike 2D maps, which provides a flat representation of the world, 3D maps offer a 1:1 correspondence with the real world. These maps can be built and represented in a multitude of ways. Digital elevation models are used for topography, photogrammetry is used on aerial/satellite imagery for 3D modeling cityscapes [1], and sensors such as LiDARs and stereo cameras are used in generating feature-based 3D maps.

Feature-based 3D maps are particularly relevant because they are extensively used in localization and pose estimation tasks in robotics, autonomous driving, and augmented reality. This is primarily because feature-based 3D point clouds capture points from the surrounding with high precision and they focus specifically on features belonging to objects in the environment such as buildings, vehicles, trees, etc. These features capture the geometry, texture, and color of objects which, when chosen with care, are robust to external factors such as lightning conditions and viewing angles. This allows a detailed representation of tightly packed environment such as city streets.

2.1.1 3D Map Density

Spatial information regarding a 3D space can either be represented using a dense 3D map or a sparse 3D map. The large amount of data points in a dense 3D map captures fine-grained structural details and represents the actual object with much higher accuracy. However, mobile systems have limited computational resources so operating on dense point clouds can be a chal-

lenging task. Moreover, the bandwidth required to transfer dense 3D maps in real-time exceeds the capabilities of current mobile network speeds. In contrast, a sparse 3D map can be used to represent the same 3D environment with significantly fewer data points. Sparse 3D maps by themselves cannot be used to infer structural properties about an object due to the low number of samples belonging to one object. Instead, it focuses on key landmarks on objects such as edges, corners etc. While the accuracy of sparse 3D maps in representing the actual object is considerably lower than dense 3D maps, they are better suited for systems with limited compute. Moreover, the reduced data requirements of sparse 3D maps make them more practical for real-time applications that interface with existing mobile network infrastructures.

2.1.2 Visual Simultaneous Localization and Mapping

Visual Simultaneous Localization and Mapping (vSLAM) is a technique used in 3D mapping and navigation where a system builds a map of an unknown environment while simultaneously tracking the position of the sensor using only visual input [26] [17]. vSLAM algorithms can be divided into feature-based methods and direct methods.

- Feature-based vSLAM: The system relies on the detection and tracking of handcrafted feature descriptors which allows the system to extract key landmarks from objects in the scene [26]. Using the feature correspondences, the system estimates the camera's position and orientation within the environment.
- Direct vSLAM: Instead of relying on handcrafted features, the system relies on pixel density or pixel intensity from the input image for tracking and mapping [26]. It does not require features to be pre-defined and matched.

While direct vSLAM can leverage all the information present in the input image, it can be computationally demanding and slight changes in lightning can yield different results. In contrast, feature-based vSLAM work only on select features from the input image and the features are robust to lighting changes.

2.1.3 Oriented FAST and Rotated BRIEF

Oriented FAST and Rotated BRIEF (ORB) features are a type of handcrafted image feature descriptor which is invariant to rotation and resistant

to noise [21]. It was created as a faster alternative to *Scale-Invariant Feature Transform* (SIFT) features while still maintaining the same performance in most situations. ORB uses the *Feature from Accelerated Segment Test* (FAST) algorithm for detecting key-points in real-time and those key-points are described using *Binary Robust Independent Elementary Feature* (BRIEF) descriptors. In essence, FAST provides the necessary computational properties for real-time key-point detection while BRIEF generates a binary string encoding for those key-points. On top of this, ORB introduces an orientation component which makes key-point features robust to rotational changes.

2.1.4 ORB-SLAM

ORB-SLAM is an example of a vSLAM algorithm uses ORB features for mapping and localization [18]. At each camera frame, ORB-SLAM extracts ORB features from visual landmarks (Fig. 2.1 left). It then matches these features across consecutive frames to estimate the camera's motion between frames. As the camera moves through the environment it builds a 3D map of the environment (Fig. 2.1 right) by adding and pruning new features collected from the environment. ORB-SLAM also has a concept called loop closure detection which allows the system to determine if camera is revisiting a previously mapped area. In such a case, it is able to account for any deviations caused by drifting features to maintain a robust feature set.

Figure 2.1: ORB feature extraction (left) and top down view of sparse 3D map (right)

2.2 Instance-level Segmentation

Instance-level segmentation is one step beyond semantic segmentation in that it not only categorizes each pixel into fixed class labels but also differentiates between objects belonging to the same category (Fig. 2.2). Instance-level segmentation involves first detecting objects in the image and designating their respective class labels followed by precisely segmenting each individual object such that all instances of the same object are distinct. A common and effective approach for instance-level segmentation is using deep learning, specifically Mask R-CNN.

Figure 2.2: Instance-level segmentation. Image source: Facebook Research. Accessed via https://github.com/facebookresearch/detectron2

2.2.1 Mask R-CNN

Mask R-CNN builds on the faster R-CNN framework by adding a branch that predicts individual object masks in addition to the bounding box. The seminal work conducted in [14] describes the exact architecture of a Mask R-CNN framework. In essence, Mask R-CNN retains the two stage process:

1. Region Proposal Network (RPN) is the first stage of Mask R-CNN and it

is identical to Faster R-CNN and it involves generating potential bounding boxes of objects of interest. The input image is passed through a series of convolutional and pooling layers to produce a feature map. Then, using a sliding window RPN generates anchor boxes with different aspect ratios. Then RPN assigns a score to each anchor box corresponding to the likeliness that an object is present within the anchor boxes. Then, non-maximal suppression is used to retain only the highest confidence bounding boxes.

2. The second stage involves detecting bounding boxes, assigning class labels, and predicting binary masks for each region of interest (RoI) in parallel [14]. During training, Mask R-CNN uses a multi-task loss function that trains classification, bounding-box prediction, and mask prediction jointly on each of the sampled RoI.

2.2.2 R101-FPN Architecture Details

The R101-FPN model uses a Residual Network + Feature Pyramid Network (ResNet+FPN) backbone with standard convolutional and fully connected heads for mask and box prediction tasks respectively [29]. The FPN takes a three channel BGR image along with its height and width (BGR, H, W) as input and outputs several feature maps at different scales because of the lateral connections in the FPN. The ResNet is contained within the FPN and it contains stem blocks and bottleneck blocks. The stem block is responsible for down-sampling the input using convolutions and max-pooling. The bottleneck blocks contain convolution layers whose kernel sizes are 1x1 and 3x3 followed by batch normalization and reLU activation. Some of these bottleneck blocks contain skip connections allowing much deeper networks to be trained without vanishing gradients.

2.3 Multi-Object Tracking

Multi-Object Tracking (MOT) is the task of identifying each object's trajectory across multiple video frames and maintaining its unique identity. MOT is generally comprised of the following key aspects:

1. Object Detection: Before any tracking can be performed, all the objects of interest in a scene must be extracted. This is generally performed using object detection neural networks such as Faster R-CNN.

2. Object Association: After the bounding boxes for objects have been acquired, the next step is to compare bounding boxes over subsequent video frames and perform a data association task to identify bounding boxes that belong to the same object.

2.3.1 Simple, Online, and Real-Time Tracking

Simple, Online, and Real-Time Tracking (SORT) is a MOT algorithm designed to combat the combinatorial complexity of traditional MOT through the usage of approximations during data association [8]. Due to its simplistic approach, it is designed to be used in real-time scenarios.

SORT capitalizes on the availability of fast object detection frameworks such as Faster RCNN and utilizes the detections produced by these models for its input. Using Kalman filter, SORT approximates the position, velocity, and inter-frame displacement of each detection in the frame. Then, it employs the Hungarian algorithm to associate the tracked objects from the previous frame to the detection in the new frame using their estimated displacements. Based on this process, SORT is able to assign unique and consistent identities to new objects that enter the scene. Additionally, it also eliminates identities for objects that exit the scene.

2.3.2 Kalman Filter

Kalman filter is a set of mathematical equations that solves the least-squares method recursively and efficiently [28]. It is used to estimate the internal state of a dynamical systems using past observations and current measurements with a possibility of noisy instruments. A Kalman filter consists of two stages - the prediction step, and the update step. In the prediction step, the past observations are run through the idealized version of the dynamical system to produce a predicted state. Then, in the update step, the new measurements with potential noise are incorporated into the predicted state to account for the uncertainty in measurements of the instruments.

2.3.3 Hungarian Algorithm

The Hungarian algorithm is often used to perform data association by configuring the problem as a bipartite matching graph. The two sets of bounding boxes of subsequent frames are assigned to the two sides of a bipartite matching graph and weighted connections are made between the two sides where the weight corresponds to the cost of association between the two bounding

boxes. The Hungarian algorithm is then responsible for finding the optimal assignment that minimizes the cost of association.

2.4 R-Trees

R-trees are an efficient method for indexing spatial data using dynamic index structures [13]. This structure corresponds to a height balanced tree where the dimensions of each child node is encompassed by its parent node (Fig. 2.3). R-trees enable fast spatial queries by minimizing the number of nodes that need to be visited in the search space [13].

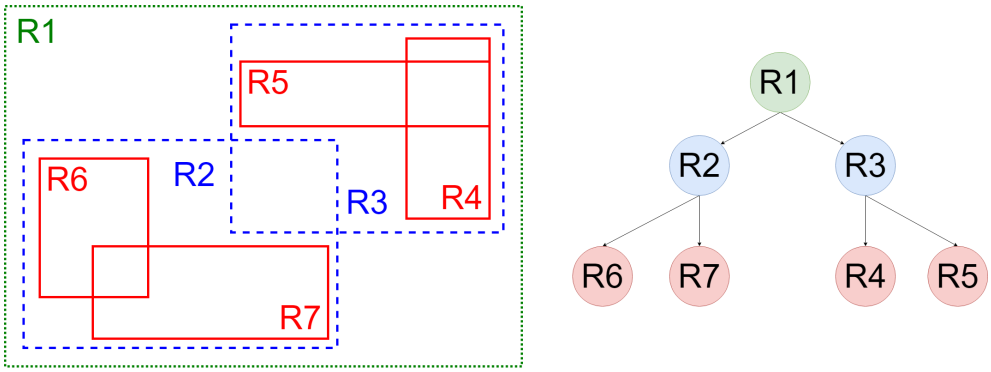


Figure 2.3: An example of a set of rectangles being represented with an r-tree

Autowaze will utilize the R-tree structure during the parking spot occupancy detection step which will require an efficient way to find the set of bounding boxes that intersect.

Chapter 3

Methodology

3.1 Overview

Autowaze consists of a network of vehicle nodes and a centralized cloud service (Fig. 3.1). Each vehicle node is equipped with stereo cameras for perception, on-board compute resources, and wireless connectivity. The vehicle nodes and the cloud nodes share a sparse 3D map (m) of the environment. Additionally, the cloud service also has a pre-annotated map of all parking spots in the area.

As the vehicle nodes navigate through the street, it collects various features from camera view (c_v) and creates a semantically enriched map of the surrounding. The system then compares (c_v) with the contents of the stored 3D map (m) using a robust feature matching algorithm. This generates two subsets of points: a) features present in (c_v) but not in (m) and b) features present in (m) but not in (c_v). These points represent the *semantic map diff* which is uploaded to the cloud service (Fig. 3.1 Cloud Upload).

At the cloud service, it updates the stored 3D map using the *semantic map diff*. Following this, it infers parking spot occupancy by estimating individual vehicle bounding boxes and comparing it with a 2D parking spot map of the region. The locations of vacant parking spots and the updates to the 3D map are broadcast back to the user-base where vehicle nodes incorporate the map updates into their stored 3D maps (Fig. 3.1 Vehicle Updates).

3.2 On-Vehicle Operations

The input to the vehicle nodes is a pair of stereo images (Fig. 3.2a.) and a stored 3D map of the environment (Fig. 3.2 b.). Autowaze performs segmentation with multi-object tracking (Fig. 3.2 c.) and feature extraction with

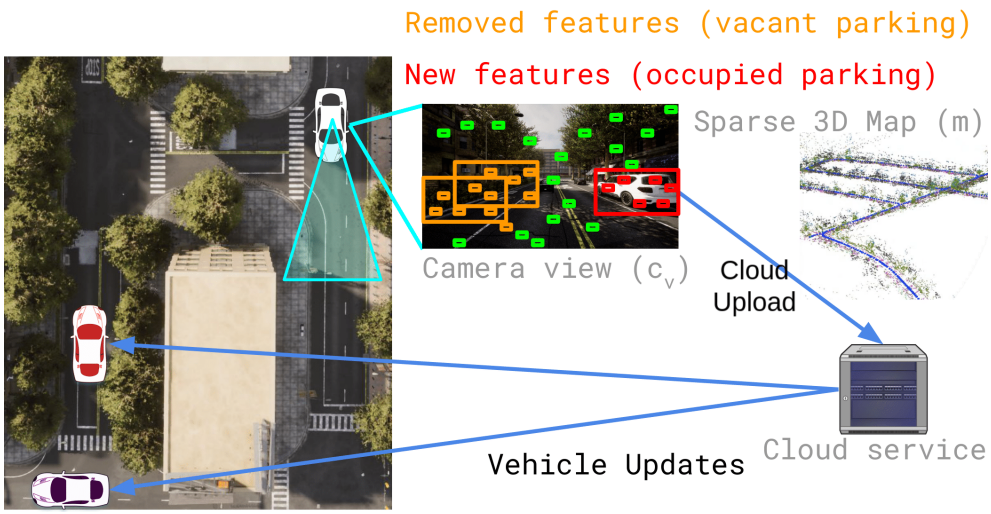


Figure 3.1: High-level system architecture

depth estimation (Fig. 3.2 d.) in parallel on a frame by frame basis. We perform merge the outputs of (Fig. 3.2 c.) and (Fig. 3.2 d.) to formulate map features, which are *camera features* containing 3D positions augmented with class labels and instance IDs from segmentation and multi-object tracking. Then these features are compared with the features of the stored 3D map to find differences in the scene over different time-windowed snapshots of the environment.

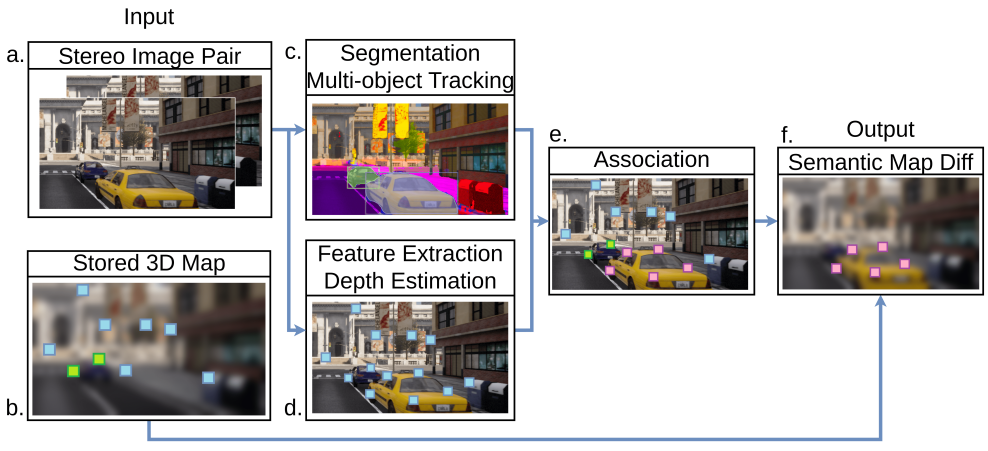


Figure 3.2: Vehicle Operations Pipeline

3.2.1 Perception

Perception takes place at the vehicle nodes using stereo cameras. Stereo vision is a common perception instrument that allows depth estimation using triangulation. Stereo cameras consist of two cameras mounted in a fixed configuration with a known short separation between the two cameras. Before sensible data can be captured, the cameras need to be calibrated and their intrinsic and extrinsic parameters must be determined. Then, as the vehicle drives through the street it captures a video sequences containing left (Fig. 3.3a) and right (Fig. 3.3b) frames. These frames are used as inputs for the rest of the system.

(a) Left Stereo Image

(b) Right Stereo Image

Figure 3.3: Stereo Image Pair

3.2.2 Mapping and Robust Feature Matching

At every stereo camera frame, Autowaze's underlying visual SLAM algorithm extracts ORB features [21] from the stereo camera images (Fig. 3.3). Using the stereo matching algorithm described in [19], Autowaze estimates the position of each feature relative to the stereo camera. These features are called *camera features* and currently they only consist of a 3D position and a feature signature.

3.2.3 Classifying Stereo Camera Image

On the same input frames, Autowaze performs instance-level segmentation to acquire object detections. Object detections are classified into several categories: pedestrians, traffic lights, stop signs, cars, trucks, etc. Each detection has the following properties:

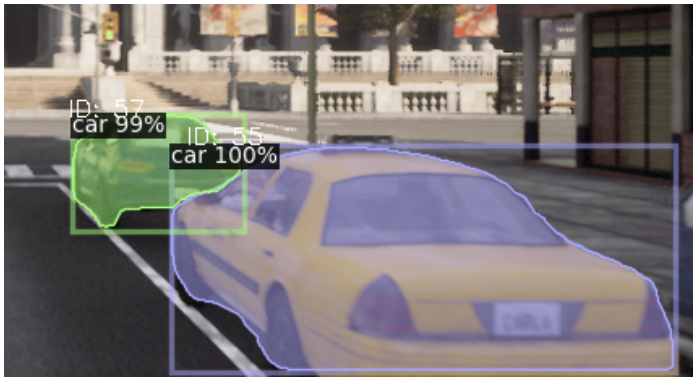
1. Position (x, y) and dimension $(width, height)$ of bounding box in the frame
2. Class label that identifies the category of the object
3. Confidence score
4. Instance ID that differentiates objects of the same class label
5. Binary mask in form of a matrix with the same dimensions as the input frame where each cell represents a pixel in the input frame. Here, cell values that are *True* represent pixels that are part of the detected object and cell values that are *False* represent pixels that do not belong to the detected object.

Autowaze filters detections based on confidence score and retains only those detections that have a high confidence for their assigned class. Instance segmentation works on a frame by frame basis. This means that an instance ID for one object is not consistent over subsequent frames. As such, the instance ID for the same object could be different in two frames. To consolidate the instance IDs, Autowaze performs multi-object tracking.

3.2.4 Multi-object Tracking

Tracking has to be performed in real-time and must only depend on information collected from past frames. In an effort to keep the computational load low, a lightweight multi-object tracker SORT is instantiated during start-up. As detection are made using the instance segmentation model, the resulting bounding boxes for each frame are passed onto the tracker. At each frame, SORT approximates the inter-frame displacement of the vehicle bounding boxes and using intersection-over-union as an association metric, associates each vehicle to their corresponding bounding boxes in the next frame. As new vehicles appear in the frame, SORT assigns unique identifiers to these vehicles. On the other hand, when vehicles go out of frame, the identifier associated with those vehicles are removed. This ensures that at any given time, the same

ID does not belong to different vehicles (Fig. 3.4). The unique identifiers that multi-object tracker assigns to vehicles are positive integers.



(a) Frame 1

(b) Frame 2

Figure 3.4: Consistent Instance IDs over different frames

3.2.5 Association

After both feature matching and multi-object tracking operations are complete, Autowaze associates the *camera features* with the class label and instance IDs. This process is simple since both *camera features* as well as instance IDs can be mapped back to their 2D pixel positions on the stereo camera frame. The *camera features* are by default captured from 2D stereo frames before being transformed into 3D features using triangulation so this transformation is straightforward. On the other hand, for instance IDs we can use the fine-grained vehicle masks generated by the instance-level segmentation

process to map each pixel to its corresponding instance ID.

3.2.6 Semantic Map Diff

During the feature matching process using visual SLAM, the algorithm can determine which features from the (c_v) matched with which features in the (m). Using this information we can infer the following:

1. If a feature is present in (c_v) and it is also present in (m), then it is a persistent feature (Fig. 3.5 Green).
2. If a feature is present in (c_v) but it is not present in (m), then it is a new feature (Fig. 3.5 Blue).
3. If a feature is not present in (c_v) but it is present in (m), then it is a removed feature (Fig. 3.5 Red).

The *semantic map diff*, is a subset of the sparse 3D map that contains the entire set of *new features* and *removed features*.

3.3 Cloud Operations

The cloud-based processing component receives the *semantic map diff* from the vehicle nodes which it uses to update the stored 3D map, estimate vehicle bounding boxes and detect parking spot occupancy (Fig. 3.6). These updates are then broadcast back to the vehicle nodes.

3.3.1 Map Diff Integration

When the cloud service receives the *map diff* from the vehicle nodes, it integrates it into the existing stored 3D map. *New features* are inserted into the 3D map while *removed features* are removed from the 3D map. Since both the cloud service and the vehicle nodes share the same 3D map, this process does not require any coordinate transformations.

3.3.2 InstanceID Based Clustering

A common solution to organizing unstructured data such as a 3D point cloud is clustering. By clustering 3D points into distinct groups, it can be reasonable to assume that points closer to each other originate from the same object and thus belong to the same cluster. This can be useful in partitioning the scene

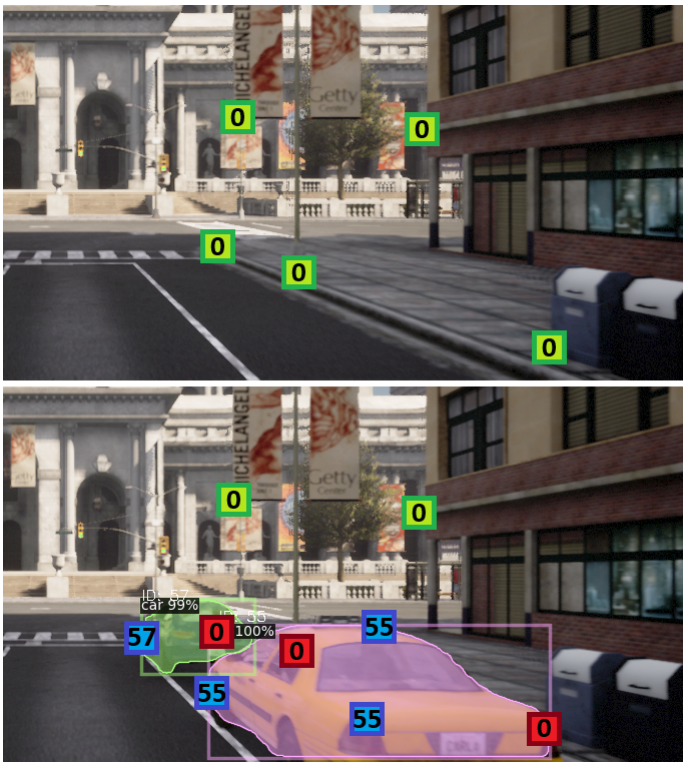


Figure 3.5: Semantic map diff over two snapshots of the environment

into distinct regions such as roadways, buildings, sidewalk etc. However, the low density of points present in a sparse 3D map coupled with the challenges of densely packed objects such as parked vehicles can cause standard clustering techniques to break down and produce erroneous results. The clusters may not align with the objects in the scene and any attempt made at inferring traffic analytics from these clusters will be marred with inconsistencies.

To address these issues, Autowaze proposes the use of semantically enriched sparse 3D maps for scene understanding. Instead of relying purely on density-based clustering to partition the point cloud, Autowaze filters the stored 3D map to only contain vehicles using their class labels. Then, the labels and vehicle IDs are used to partition the point cloud into distinct groupings. This approach allows fine-grained segmentation of the point cloud where the groupings align much closer with the actual objects in the scene. Even in tightly packed scenarios, the use of this technique will ensure that points originating from two closely parked vehicles are not conflated resulting in a robust

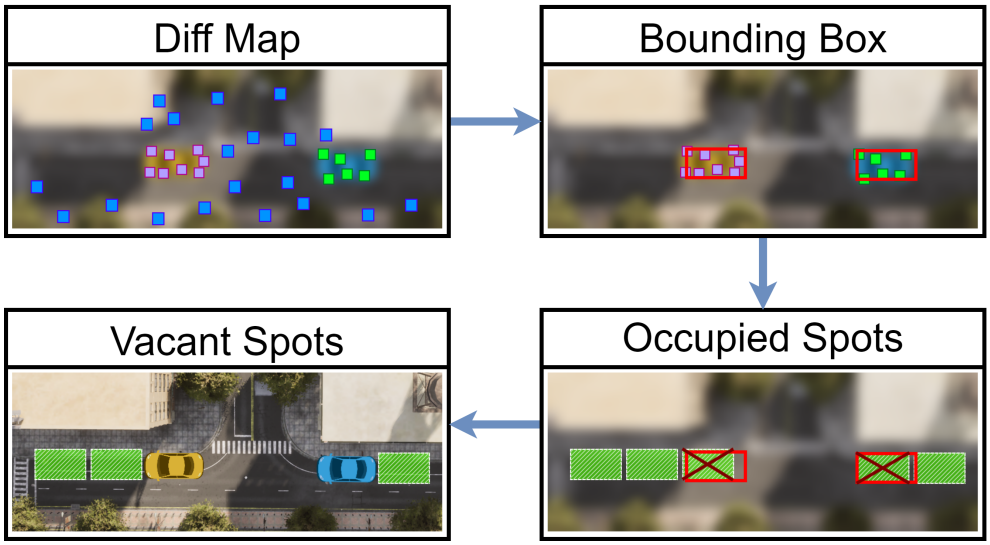


Figure 3.6: Cloud Operations Overview

foundation for making inferences on traffic events.

3.3.3 Outlier Removal

To remove outliers from the grouped point cloud, Autowaze employs a neighborhood outlier removal method. This process helps to identify and remove map points that are too far away from the parked vehicle's current position, ensuring a more accurate and reliable representation of points emerging from the vehicle. For each map points in the map diff a local neighborhood is defined using the ID-based grouping. Any point in a grouping that is not close enough to a sufficient number of neighbors of the same group are eliminated from the group. The key advantage of neighborhood outlier removal is that it allows us to take into account the shape and structure of different types of vehicles rather than rely on global statistics.

3.3.4 Bounding Box Estimation

Autowaze employs a simple implementation of bounding box estimation. Essentially, once the point cloud is grouped based on their unique identifiers, it computes the centroid for each cluster. This centroid acts as the center of mass for the vehicle. Then, originating outwards from the centroid, Autowaze

draws a bounding box that is of a fixed size and roughly equals the size of an average vehicle [2].

3.3.5 Vacant Parking Spot Detection

Autowaze detects parking spot occupancy using a two pass method. On the first pass, it creates an r-tree using the 2D parking occupancy map where the leaf nodes are parking spot dimensions and each parent node encompasses the combined dimensions of the leaf nodes. Using this r-tree, Autowaze efficiently extracts vehicle bounding boxes that are overlapped with parking spots on the map. It iterates through the intersection set and determines the bounding box containing the highest amount of overlap with the parking spot. This bounding box is then associated to the parking spot and the parking spot is considered occupied (Fig. 3.7 First Pass). Although this removes a good chunk of occupied parking spots, the inherent uncertainty with SLAM can cause false positives in this process.

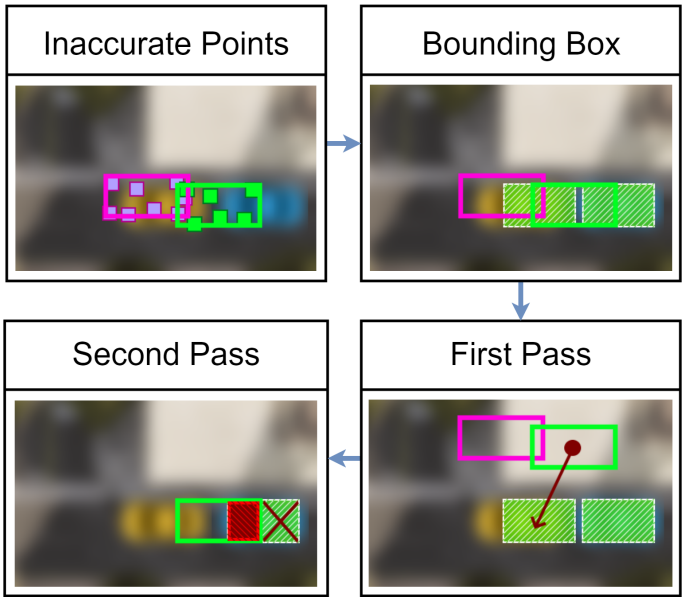


Figure 3.7: Parking Spot Occupancy Detection

SLAM drift is a phenomenon that can cause inaccuracies in the position of features in the environment. Features emerging from vehicles may be offset from their true position in the world (Fig. 3.7 Inaccurate Points). In such a case, the incorrect vehicle may be associated with a parking spot in the first

pass. Without prior knowledge of how and when the SLAM drif occured, the position of these features cannot be easily rectified. To reduce errors caused due to SLAM drift, a second pass must be conducted where the goal is to minimize false positives by culling any parking spot that contains a significant overlap with any bounding box.

During the second pass, another r-tree is built but this time using the bounding boxes in the leaf-nodes. Then, for each remaining vacant parking spot, it finds the set of bounding boxes that share an overlap. If the overlap is significant, it indicates that some SLAM drift may have occurred and these parking spots are also considered occupied (Fig.3.7 Second Pass).

3.3.6 Transmission To Vehicle Nodes

The output of the vacant parking spot detection is a list of rectangles each containing the following properties: x-coordinate, y-coordinate, width, height. These rectangles can be packaged up into a JSON array of JSON objects containing the key/values pairs for each of the aforementioned properties. The JSON is broadcast from the cloud service to the vehicle nodes in addition to any map updates.

Chapter 4

Evaluation

4.0.1 System Implementation

The on-vehicle operations involving extracting features from the stereo camera frames using a modified version of ORB-SLAM2 [19] and creating the *semantic map diff* are written in C++. The detection and tracking of vehicles in the scene is written in python with the help of the Detectron2 library [29] and the SORT source code [8]. The specific model that we use for instance-level segmentation is a pre-trained model called R101-FPN trained on the COCO dataset available via the Detectron2 model zoo [29]. Image transformations and vehicle mask manipulation was carried out using OpenCV [9]. Operations involving point-cloud manipulation or visualization were made possible with the help of Point Cloud Library (PCL) [22] and Matplotlib [27].

In this section, we evaluate the performance of the system by conducting a series of experiments that determine the accuracy of the results, application benefits, and the responsiveness of the Autowaze. The end-to-end experiments were conducted on computer equipped with an AMD Ryzen 7 7800X3D processor with 16 CPUs running at approximately 4.2GHz with 32 GB DDR5 RAM and an NVIDIA RTX 4080 GPU containing 9728 CUDA cores and capable of 780 Tera Operations per Second.

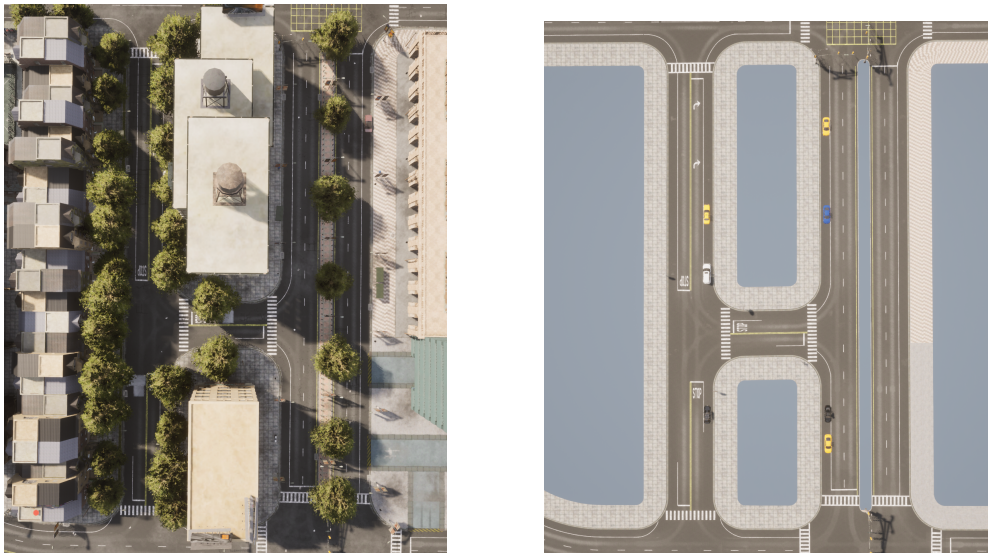
4.1 Experiment Setup

4.1.1 CarLA Simulator

We used a photo-realistic autonomous driving simulator to collect about 50 traces. We isolated two streets in the "Town 10" map provided by CarLA and spawned parked vehicles throughout the street parking zones on the side (Fig.

4.1). We then equipped the ego-vehicle with stereo cameras to extract the left and the right images. Finally, we established way-points for the ego-vehicle to follow and collect key-points from the two blocks to formulate the 3D point cloud.

In each scenario, there are a total of 15 fixed spots that the parked vehicles could occupy. During each pass through, we select a random amount of spots to be occupied and then start recording. The model, make, and color of the parked vehicles is randomly selected excluding some over-sized vehicles.



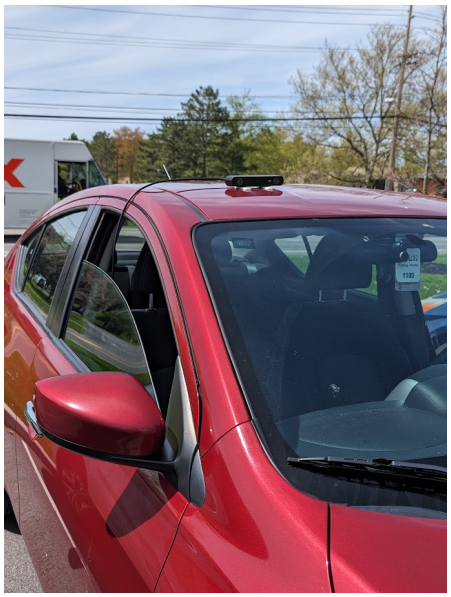
(a) Top-down view of "Town 10" with buildings and foliage layer active

(b) Top-down view of "Town 10" with buildings and foliage layer inactive

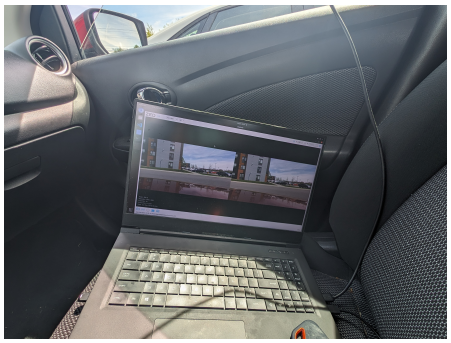
Figure 4.1: CarLA map experiment setup

4.1.2 Real-world

In addition to the simulator, we also tested Autowaze in the real-world by mounting a vehicle with a ZED 2i [30] stereo camera (Fig. 4.2a) and connecting it to a gaming laptop within the vehicle (Fig. 4.2b). Using this, we collected 20 traces on three streets in the Rochester metropolitan area (Gold St., Langslow St., and Stewart St.) including a few traces from the Rochester Institute of Technology, Parking Lot B during different days and lighting conditions.



(a) Mounted ZED 2i stereo camera



(b) On-board compute resource

Figure 4.2: Real-world experiment setup

4.1.3 Accuracy Metric

To assess the performance of the parking spot detection, we computed the precision, recall, and accuracy metrics. The ground truth for each trace was manually annotated using the Carla environment.

Precision and recall are defined as following:

- $\text{Precision} = \dfrac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$
- $\text{Recall} = \dfrac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$
- $\text{Accuracy} = \dfrac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}}$

where,

- True Positive (TP) represents an accurately predicted vacant spot
- True Negative (TN) represents an accurately predicted occupied spot

- False Positive (FP) represents an occupied spot being predicted as a vacant spot
- False Negative (FN) represents a vacant spot being predicted as an occupied spot

4.1.4 Baseline Implementations

To evaluate the accuracy of our system, we compare it against two baselines. The first baseline replicates visual SLAM [19]. In this, the vehicle sends a feature-based 3D map to the cloud service, which then determines parking spot occupancy by clustering points using DBSCAN [24]. The second baseline replicates CarMap [7]. In this, the vehicle sends a map diff to the cloud service which integrates it into the base map. Then, the cloud service filters out points belonging to vehicles from the updated map and runs DBSCAN on them.

4.2 Parking Spot Detection Accuracy

In the experiments conducted in CarLA (Table. 4.1), the precision for Autowaze greatly outperformed both the baseline implementations. Autowaze had a precision of 0.87 meaning that out of all the parking spots that Autowaze predicted as vacant, 87% of them were truly vacant. This higher precision compared to ORB-SLAM2 and CarMap can be attributed to the way Autwaze groups the point cloud based on instanceIDs rather than relying on statistical clustering. Autowaze also makes use a the two pass method for minizing false positive results introduced by SLAM drift. Because ORB-SLAM2 and CarMap have no mechanisms to deal with these errors, they have a tendency to falsely mark occupied spots as vacant.

The baseline implementations had a higher recall than Autowaze because ORB-SLAM2 and CarMap over-classify vacant spots, leading to reduced number of false negatives. However, the trade-off of this is that it also increases the number of false positives. Autowaze on the other hand, manages to strike a good balance between precision and recall. This balance is important for detection systems and Autowaze not only achieves this, but it also has an overall higher accuracy of 0.89.

A similar trend was observed in the real-world tests (Table. 4.2) with Autowaze outperforming the baseline in both precision and accuracy. Autowaze was able to achieve a precision of 0.93, a recall of 0.82 and an accuracy of 0.84 further reinforcing its effectiveness over the baseline approaches.

Approach	Precision	Recall	Accuracy
ORB-SLAM2 [19]	0.67	0.98	0.76
CarMap [7]	0.77	0.88	0.82
Autowaze	0.87	0.91	0.89

Table 4.1: Parking spot occupancy detection (CarLA)

Approach	Precision	Recall	Accuracy
ORB-SLAM2 [19]	0.78	0.87	0.75
Autowaze	0.93	0.82	0.84

Table 4.2: Parking spot occupancy detection (real-world)

4.3 System Latency

To measure Autowaze's end-to-end latency, we profiled the vehicle and cloud operations in the simulated CarLA dataset.

Figure. 4.3 (left), shows the average latency of the tracking (SLAM), instance-level segmentation (Seg), multi-object tracking (MoT), and semantic map diff (Diff). Based on these results we can see that tracking is the most expensive operation taking on average 41ms. Tracking is performed in the SLAM algorithm which is not an additional cost introduced by Autowaze. Segmentation and multi-object tracking happen in parallel and with the combined average run-time being less than the tracking operation, the tracking operation dominates the run-time of the vehicle operations. The only additional cost on-top of SLAM would be the generation of the semantic map diff which takes 9ms on average. This means that Autowaze is able to generate a semantic map diff in about 50ms. This shows that if the stereo camera is running at 10fps, the system is capable keeping up with the camera.

Figure. 4.3 (right) reports the latency of the cloud operations including the integration of the semantic map diff as well as the detection of parking occupancy. In these experiments, we had Autowaze capture semantic map diffs every 5 seconds on a 10fps stereo camera which equal to 50 frames of data. On average, it took the cloud service 47ms to perform both map diff integration and parking occupancy detection.

Since the vehicle operations can run in about 50ms and the cloud service can run in 47ms, Autowaze can detect and report vacant parking spots in approximate 100ms which is a good benchmark for most vehicle assistance systems.

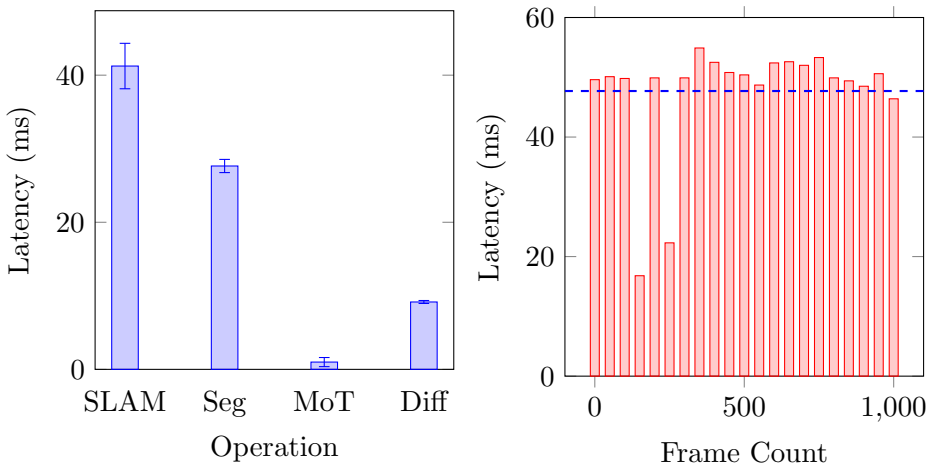


Figure 4.3: End-to-end latency of Autowaze's vehicle operations (left) and cloud operations (right)

4.4 Empirical Analysis of Applicational Benefits

4.4.1 Search Time Reduction

In order to quantify the potential reduction in search time per driver, we modified a satellite image of Parking Lot D at the Rochester Institute of technology to include additional vehicles and an annotated parking spot as shown in Fig. 4.4. The captured section of the parking lot is approximately 80 meters long and 30 meters wide. Starting at the entrance of the parking lot, given that a vehicle is travelling at $10mph$, a vehicle without Autowaze will travel a maximum of 195 meters to find the vacant parking spot which will take 43.65 seconds. This is because the driver has no knowledge of the whereabouts of the vacant spot and will perform an exhaustive search of the parking lot to find the vacant parking spot. This is indicated by the *red path* in Fig. 4.4. In contrast, a vehicle with Autowaze will always use the shortest path to travel to the vacant parking spot because it has prior knowledge of the location of the vacant parking spot. This is indicated by the *green path* in Fig. 4.4. As such, it will only travel about 35 meters which will take 7.83 seconds. That is a reduction of 82.1% in search time for this particular parking scenario.

4.4.2 Fuel Efficiency and Cost Savings

Referring back to Parking Lot D given in Fig. 4.4, if the average mileage given by the vehicle is 25 miles per gallon (*mpg*), and the cost of gas price is $3.50 per gallon, then the vehicle without Autowaze will expend at-most 0.0048 gallons to travel 195 meters and the search will cost $0.0168. On the other hand, the vehicle with Autowaze will only expend 0.00088 gallons to complete the same search and it will only cost $0.0031. This improves both the fuel efficiency and cost savings by about 81%.

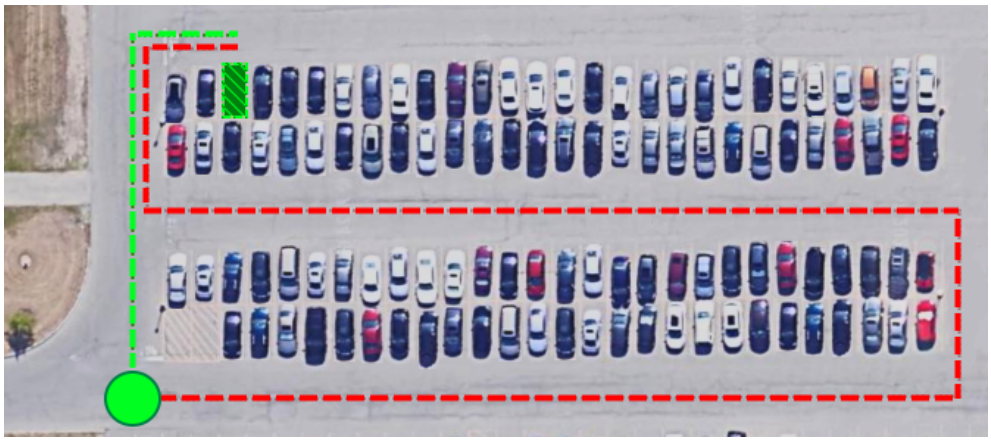


Figure 4.4: Modified satellite imagery of Parking Lot D at RIT [3]

4.5 Network Costs

Over a period of 5 second, for a stereo camera with 10fps, the size of the *semantic map diff* is anywhere from 1KB to 60KB. According to the Federal Communications Commission (FCC) broadband deployment report [12], this is well within the capabilities of modern broadband speeds.

Chapter 5

Conclusion and Future Work

5.1 Conclusions

The work presented in this thesis has demonstrated the design, implementation and evaluation of Autowaze, a system capable of reporting accurate parking spot occupancy status at real-time using a crowd-sourced strategy. Building on exiting methods for feature-based sparse 3D maps designed for localization tasks, Autowaze semantically enhances these maps with the help of instance-level segmentation and multi-object tracking.

Through experiments conducted both in simulation and the real-world, Autowaze is capable of determining the occupancy status of a parking spot with an accuracy of 89%. Moreover, it showcases that with the help of fine-grained traffic analytics such as the location of vacant parking spots, motorists can make smarter decisions to reduce idle time and improved fuel efficiency.

Overall, this thesis demonstrates the feasibility of extracting high-level semantics from sparse 3D maps. It also provides a solid foundation for future research to be conducted on automated traffic analytics and driver assistance domains.

5.2 Future Work

There are several areas in this thesis that are good avenues for future research. **Additional Traffic Analytics.** This thesis solely focuses on parking spot occupancy detection but in order to be a fully fledged traffic analytics system, it must also account for road closures, accidents, construction etc. A similar approach could be taken for construction where features originating from traffic cones can point towards signs of road construction.

Integration of LiDAR. While Autowaze uses stereo cameras for feature maps, using a LiDAR to map the environment would result in higher accuracy and precision. In fact, many issues that Autowaze faced are corelated to the inherent imprecision of SLAM on stereo camera images. False positives and false negatives arise when SLAM is unable to localize accurately which results in the features being offset from their true positions. Future work should focus on adapting *semantic map diff* with LiDAR point clouds instead. However, one of the main goals of Autowaze is to adapt widely existing vehicular technology to solve the problem and using LiDARs, atleast in it's current state, would defeat the purpose of the study.

Dynamic Environments. Most of the traces that were used did not have actively moving traffic. The inaccuracy of SLAM and the amount of moving objects makes this a very difficult problem to solve in the dense cities with bumper to bumper traffic. Further research is needed to investigate techniques that are capable of dealing with high amounts of moving traffic in close proximity of sensors and parking spots. This might potentially require a shift to better sensing technologies.

Bibliography

[1] Photogrammetry – history and modern uses, 2022. Accessed: May 9, 2024.

[2] Average car length: All you need to know about it, n.d. Accessed: May 9, 2024.

[3] Google earth, n.d. Accessed: May 9, 2024.

[4] New york handheld device law, n.d. Accessed: May 9, 2024.

[5] Smart parking, n.d. Accessed: May 9, 2024.

[6] Waze, n.d. Accessed: May 9, 2024.

[7] Fawad Ahmad, Hang Qiu, Ray Eells, Fan Bai, and Ramesh Govindan. CarMap: Fast 3d feature map updates for automobiles. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 1063–1081, Santa Clara, CA, February 2020. USENIX Association.

[8] 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. IEEE, 2016.

[9] Gary Bradski. The opencv library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 25(11):120–123, 2000.

[10] Pietro Carnelli, Joy Yeh, Mahesh Sooriyabandara, and Aftab Khan. Parkus: A novel vehicle parking detection system. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, pages 4650–4656, 2017.

[11] Weiling Chen and Chai Kiat Yeo. Unauthorized parking detection using deep networks at real time. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 459–463. IEEE, 2019.

[12] Federal Communications Commission. Fourteenth broadband deployment report. https://www.fcc.gov/reports-research/reports/broadband-progress-reports/fourteenth-broadband-deployment-report, 2021.

[13] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, 1984.

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[15] Inrix. The impact of parking pain in the us, uk and germany, 2017.

[16] Inrix. Inrix 2022 global traffic scorecard: London tops list as most congested city, u.s. cities inch closer, 2022.

[17] Iman Abaspur Kazerouni, Luke Fitzgerald, Gerard Dooly, and Daniel Toal. A survey of state-of-the-art on visual slam. *Expert Systems with Applications*, 205:117734, 2022.

[18] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[19] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.

[20] Sarfraz Nawaz, Christos Efstratiou, and Cecilia Mascolo. Parksense: A smartphone based sensing system for on-street parking. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 75–86, 2013.

[21] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[22] 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. IEEE, 2011.

[23] Rosario Salpietro, Luca Bedogni, Marco Di Felice, and Luciano Bononi. Park here! a smart parking system based on smartphones' embedded sensors and short range communication technologies. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 18–23. IEEE, 2015.

[24] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

[25] Leon Stenneth, Ouri Wolfson, Bo Xu, and S Yu Philip. Phonepark: Street parking using mobile phones. In *2012 IEEE 13th international conference on mobile data management*, pages 278–279. IEEE, 2012.

[26] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: A survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):1–11, 2017.

[27] Sandro Tosi. *Matplotlib for Python developers*. Packt Publishing Ltd, 2009.

[28] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.

[29] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[30] ZED Stereo Camera. `https://www.stereolabs.com/`.