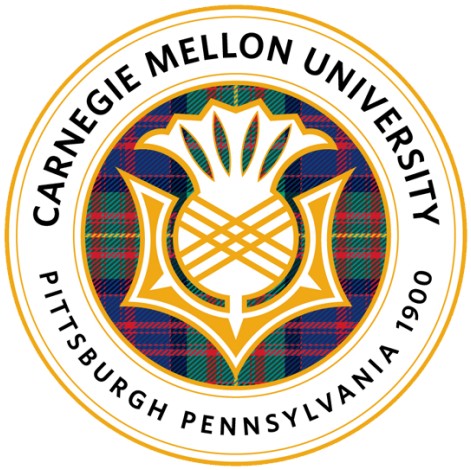
Computer Vision for Live Map Updates

Kevin Christensen
June 2019



The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee
Mahadev Satyanarayanan, Advisor
Martial Hebert, Advisor
Christoph Mertz
Padmanabhan Pillai (Intel Labs)
Junjue Wang

CMU-RI-TR-19-40

Submitted in partial fulfillment of the requirements
for the degree of Master of Science.

Keywords: Computer Vision, LiveMap, Edge Computing, Visual Change Detection

For my family.

Abstract

Real-time traffic monitoring has had widespread success via crowd-sourced GPS data. While drivers benefit from this low-level, low-latency map information, any high-level traffic data such as road closures and accidents currently have very high latency since such systems rely solely on human reporting. Increasing the detail and decreasing the latency of this information has significant value. In this work we explore the idea of real-time crowd-sourced map updates from visual data. We propose a system that uses object detection to detect hazards which are then reported via 4G LTE to a local server on the edge. This edge server aggregates the data and relays updates to other vehicles inside its zone. We call our system *LiveMap*. We demonstrate detection accuracy on hazards and characterize the system latency. We propose and develop two extensions that can improve system functionality. The first improvement is a semantic change detection pipeline, which can detect changes between image pairs to provide high-level map updates as well as enable accurate removal of stale hazards. Finally we develop a novel visual odometry algorithm to improve hazard localization.

Acknowledgments

First and foremost I would like to thank my advisors, Professor Mahadev Satyanarayanan and Professor Martial Hebert for their support and guidance during my two year journey here at Carnegie Mellon University. I could not have asked for better advisors. Their discussions and guidance have greatly influenced my research interests and passions. I would also like to thank Christoph Mertz and Padmanabhan (Babu) Pillai whose counsel was invaluable in forming my thesis.

I would also like to give a special thanks to Thomas Eiszler, who helped me annotate images and helped for every live demonstration, and more. I would also like to thank Jan Harkes, Junjue Wang, Edmond (Ziqiang) Fang, and Shilpa George for their help and for being great labmates. I also want to thank John Kozar for his assistance driving Navlab while testing the system.

Last but not least, I would like to thank research sponsors for which I am extremely grateful. This research was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0051 and by the National Science Foundation (NSF) under grant number CNS-1518865. Additional support was provided by Intel, Vodafone, Deutsche Telekom, Verizon, Crown Castle, NTT, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view(s) of their employers or the above-mentioned funding sources.

Contents

1 Introduction **1**

1.1 Contributions . . . 3

1.2 Thesis Outline . . . 3

2 Background and Related Work **5**

2.1 Deep Learning . . . 5

2.2 Network Architectures . . . 6

2.2.1 Semantic Segmentation . . . 6

2.2.2 Object Detection . . . 8

2.2.3 Instance Segmentation . . . 8

2.2.4 Panoptic Segmentation . . . 9

2.2.5 Datasets . . . 10

2.3 Loss Functions . . . 10

2.3.1 Classification . . . 10

2.3.2 Faster R-CNN . . . 11

2.4 Edge Computing . . . 11

2.5 Crowd-sourcing Applications . . . 12

2.6 Related Work . . . 13

2.6.1 LiveMap . . . 13

2.6.2 Visual Change Detection . . . 14

2.6.3 Visual Odometry . . . 15

3 LiveMap **17**

3.1 System Architecture . . . 17

3.2 Implementation . . . 19

3.2.1 Zone Cloudlet . . . 19

3.2.2 Vehicle Cloudlet . . . 20

3.2.3 Big Vehicle Cloudlet . . . 20

3.2.4 Small Vehicle Cloudlet . . . 22

3.3 Experimental Results . . . 22

3.3.1 Additional Hazards . . . 25

4 Visual Change Detection **29**
4.1 Motivation . . . 29
4.2 Problem Statement . . . 31
4.3 System Pipeline . . . 31
4.4 Network Architecture . . . 31
4.4.1 Training . . . 32
4.5 Experiments . . . 33
4.6 Discussion . . . 34
4.6.1 Failure Modes . . . 34
4.6.2 VL-CMU-CD Dataset Limitations . . . 36

5 Visual Odometry **41**
5.1 Motivation . . . 41
5.1.1 Contributions . . . 41
5.1.2 Visual Odometry vs. SLAM . . . 42
5.2 Edge-Direct Visual Odometry . . . 43
5.2.1 Overview . . . 43
5.2.2 Camera Model . . . 44
5.2.3 Camera Motion . . . 45
5.2.4 Robust Gauss-Newton on Edge Maps . . . 46
5.2.5 Optimizing over Edge Points . . . 47
5.2.6 Keyframe Selection . . . 47
5.3 Experiments . . . 47
5.3.1 Evaluation Metrics . . . 48
5.3.2 Results on the TUM RGB-D Benchmark . . . 50
5.3.3 Ablation Study . . . 50
5.4 Discussion . . . 51

6 Conclusion **53**
6.1 Future Work . . . 53

Bibliography **55**

List of Figures

1.1 Waze mobile application interface . . . 2

2.1 U-Net Architecture . . . 7
2.2 SegNet Architecture . . . 7
2.3 Faster R-CNN Architecture . . . 8
2.4 Mask R-CNN Architecture . . . 9
2.5 UPSNet Architecture . . . 9
2.6 Tiered Model of Computing Landscape . . . 12

3.1 LiveMap System Overview . . . 18
3.2 LiveMap Implementation Details . . . 19
3.3 LiveMap Example Interface for Hazards . . . 21
3.4 Big Vehicle Cloudlet . . . 21
3.5 Annotated Pothole Detections . . . 23
3.6 Bandwidth vs. Event Recall . . . 25
3.7 LiveMap Live Run Interface . . . 26
3.8 LiveMap Live Run Detection Example . . . 26
3.9 LiveMap Live Run Multiple Detection Example . . . 27
3.10 LiveMap Live Run Multiple Detection with Missing Detection Example . . . 27

4.1 VL-CMU-CD Dataset . . . 30
4.2 Change Detection System Pipeline . . . 32
4.3 Change Detection CNN Architecture . . . 33
4.4 Binary Precision-Recall Curve . . . 35
4.5 Binary ROC Curve . . . 35
4.6 Multi-class Precision-Recall Curve . . . 36
4.7 Qualitative Results on VL-CMU-CD Dataset 1/2 . . . 37
4.8 Qualitative Results on VL-CMU-CD Dataset 2/2 . . . 38
4.9 Semantic Change Detection Network Failure Modes . . . 39
4.10 VL-CMU-CD Dataset Ground Truth Errors . . . 40

5.1 Photometric Edge Error Illustration . . . 42
5.2 Edge Extraction Examples . . . 43
5.3 Edge Pyramid for Canny Edges . . . 44
5.4 Edge Residuals vs. All Residuals . . . 45

5.5 Trajectory Error with VO 49
5.6 VO Ablation Study 51
5.7 VO Tracking Performance on Various Sequences 52

List of Tables

3.1 System Latency 22
3.2 System Hazard Detection Results 24

4.1 Binary Change Detection Quantitative Results 34
4.2 Change Detection Quantitative Results Taken with Argmax Output 34

5.1 Main Visual Odometry Results 48

Chapter 1

Introduction

Over 100-million active users benefit from Waze every month [100]. This crowd-sourced application allows drivers and passengers to report road events to a collecting entity that merges information and overlays it on GPS navigation maps via the mobile Waze app [115]. Figure 1.1 shows the Waze app interface.

In 2013 Google purchased Waze for 1.15 billion dollars [100], which is an indication of the perceived value for its service. Unfortunately the benefits of Waze come at the cost of user distraction, which is known to be a major source of traffic accidents [8, 33]. Since the majority of vehicles have just the driver in the car with no passengers, Waze reports are typically made by a driver who incurs distraction in creating and submitting the Waze report. In short, the service is valuable but dangerous to not only the driver but also to nearby drivers, pedestrians, and bicyclists. With advances in computer vision and edge computing, we ask the question: "Can we have the benefits of Waze without user distraction?"

An automated Waze-like system requires sensing capabilities to detect hazards, a means to report the hazards, and an entity to synthesize hazard reports and distribute to other users. The method of sensing must be both accurate and generalizable to a number of different types of hazards. The system as a whole must be scalable to a large number of vehicles in order to efficiently crowd-source and distribute the data.

Our approach to solve this challenging problem utilizes an in-vehicle camera with an in-vehicle computer termed the *Vehicle Cloudlet*, to run computer vision algorithms to detect hazards from visual data. We use the term *cloudlet* to refer to an edge-located server that is connected via either a local area network (LAN) or wide area network (WAN) with few network hops, and is responsible for providing *edge computing*, or computational services provided at close network proximity. Hazards are then reported via 4G LTE to another cloudlet, termed the *Zone Cloudlet*. The Zone Cloudlet aggregates the data, stores it in a database, and notifies other Vehicle Cloudlets inside its zone of responsibility.

In order to detect hazards with computer vision we rely on a Convolutional Neural Network (CNN). CNNs have been driving progress across various computer vision tasks for the last decade. Hazard detection can be categorized as a type of object detection; thus we focus on object detection architectures to sense hazards. CNNs are computationally demanding, which requires the use of GPUs to efficiently run at or near real-time. Cloud or edge computing is frequently turned to in order to provide the required compute services. We make a key distinc-

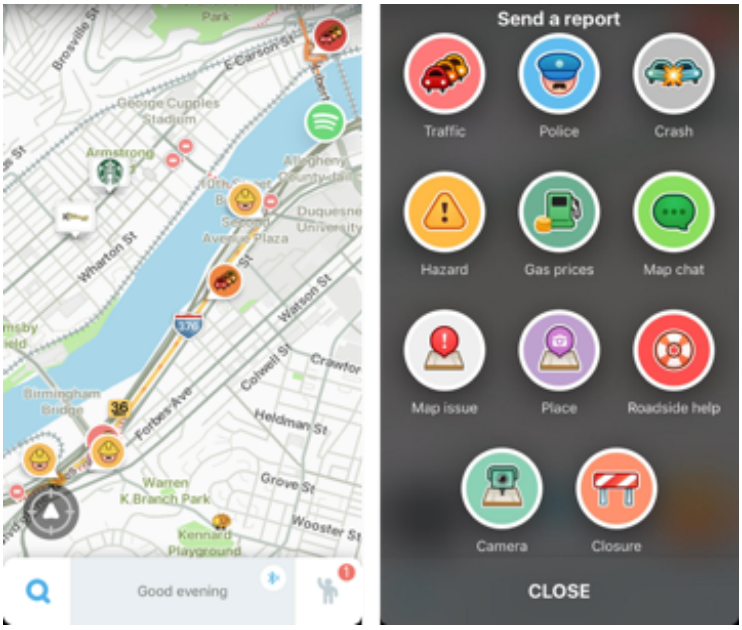


Figure 1.1: Left: A screenshot of Waze app with road events overlayed on GPS navigation map. Right: A screenshot of the Waze hazard reporting interface.

tion between cloud and edge computing. Cloud computing requires the traversal of a WAN with many network hops to access its services, whereas edge computing requires the traversal of either a LAN or WAN with few network hops to access its services.

Real-time map updates can extend beyond hazard identification as well. It can also include detection of changes to existing road maps. Such a system can be extremely useful to provide a wide range of map updates. Accurate maps are becoming increasingly important as the number of self-driving car companies grow. Autonomous vehicles currently rely on accurate maps for self localization and for detection and localization of other map features such as traffic lights. Examples of useful map updates include the ability to identify new or removed signs and buildings, construction, vehicles that are parked illegally or past expired time, or even removing hazards that have disappeared, and more. Keeping maps up-to-date is a costly yet paramount endeavor. The ability to crowd-source this information, and do so continuously has significant value. Change detection requires the use of a generalizable pipeline that can detect well-defined classes of change. We turn to the task of visual change detection to provide such functionality. We create a custom CNN architecture that is capable of detecting change at a pixel level between two aligned images.

While CNNs are useful for hazard and change localization at a pixel level, we require the use of GPS in order to localize such updates on an actual map. Visual odometry (VO) and simultaneous localization and mapping (SLAM) are examples of two such algorithms that can enable more accurate vehicle localization as well as more accurate positioning of map updates. The ability to more accurately localize hazards would enable the system to identify duplicate reports with ease, or use a consensus from multiple vehicles to establish a higher confidence of hazard detections. We create a novel visual odometry algorithm that can provide more accurate

tracking of vehicle pose as well as hazard location.

1.1 Contributions

The core of this thesis focuses on a scalable system architecture along with a suite of vision tools for accurate map updates. The contributions of this thesis are more explicitly stated as follows:

- An implementation of *LiveMap*, an automated hazard detection system.
- A semantic change detection pipeline that uses a novel Convolutional Neural Network (CNN) architecture to localize and classify change at a pixel level.
- An edge-direct visual odometry algorithm that outperforms state-of-the-art methods on public datasets.

1.2 Thesis Outline

This thesis is organized as follows. In Chapter 3 we present *LiveMap* [13]. We focus on the architecture and development of a prototype for such a system, and demonstrate it detecting hazards such as potholes which are frequently found locally for testing. We extend its capabilities to be able to detect traffic cones. Since the system utilizes 4G LTE and requires processing of high bandwidth data, we quantify the bandwidth saved by moving compute inside the vehicle. In Chapter 4 we present our pipeline for semantic change detection, and benchmark its performance compared to prior work. Finally, in Chapter 5 we present an edge-direct visual odometry algorithm can enable more accurate vehicle tracking which would provide more accurate hazard localization and map updates.

Chapter 2

Background and Related Work

The crowd-sourcing of meaningful context from images requires both computer vision as well as system design. An overview of relevant topics in computer vision and cloud and edge computing is provided up until the point of writing this thesis. Advancements in computer vision have brought about the need for cloud and edge computing to deliver the required computational resources for such computationally demanding applications.

2.1 Deep Learning

The field of computer vision and machine learning has advanced tremendously with the help of deep learning. Many areas within computer vision have seen great progress due to the advent of Convolutional Neural Networks (CNNs). CNNs have enabled large performance gains across several important tasks.

Neural networks are not new, and have been explored decades ago, such as in handwritten zip code recognition [62] and in other works [63, 94]. However, the size of linear layers has been prohibitively expensive in terms of memory and compute for successive layers. The solution to this issue was a combination of advances in GPU programming combined with advances in CNN architectures. In 2012, AlexNet [58] kicked off the deep learning era by surpassing previous methods on the ImageNet Large Scale Visual Recognition Challenge (ILSVR) [20, 89]. Since then, CNNs have exploded in popularity for visual tasks which require complex information extracted from images. In CNNs, hierarchical features are extracted through successive convolutional layers. These features are then used as input into either a linear layer for regression tasks, or upsampled or passed into transposed convolutional layers for pixel level classification tasks. CNNs have achieved massive success is tasks such as: 1) object detection in which a bounding box is proposed for an object and simultaneously classified, 2) semantic segmentation in which an image is classified at a pixel-level, 3) change detection in which pixels are highlighted as change across two images of the same scene, 4) instance segmentation where objects are localized with pixel-level masks and classified, and very recently 5) panoptic segmentation [54] which is a combination of semantic and instance segmentation. The use of CNNs has also been extended to use on 3D data for tasks such as 3D Classification, 3D Segmentation, and Point Cloud Completion, as well as for temporal video sequences such as action recognition which

remains a difficult task even for deep CNNs. CNNs are very successful across tasks because they can be trained efficiently with gradient descent, and compared to successive linear layers they decrease the number of required parameters through the use of convolution.

When selecting a network architecture for a task and especially when creating one from scratch, one must be concerned with network structure as well as weight initialization, activation functions, optimizers, and hyper-parameter tuning. The field has been moving rapidly, and there have been many advancements across all of these areas. Much of the success of deep learning is due to the automated process of learning. Stochastic gradient descent [7] has enabled fast training on very large datasets of millions of filter parameters for feature extraction. Such automated features have replaced hand-crafted features that were commonly used prior, such as Histogram of Oriented Gradients (HOG) features, or Scale-Invariant Feature Transform (SIFT) [68].

The convergence of stochastic gradient descent can often be problematic, as large gradients cause convergence to be erratic which leads to poorly converged solutions. Mini-batch gradient descent has emerged in response, which simultaneously provides more robust convergence properties as well as reduced training time due to less frequent parameter updates [64]. Even with mini-batch gradient descent, optimization of CNNs is tricky and often difficult. The Adam optimizer [53] is one such first-order gradient descent method that has seen widespread success in deep learning. RMSProp [108] and Adagrad [22] are two other commonly used optimizers. Research on more robust optimizers is still ongoing. To complicate matters, network architectures heavily impact the choice of optimizer and learning rate.

Gradient descent is well-studied and known to be sensitive to initialization. In [37] the authors explored the effects of weight initialization, which led to the commonly used 'Xavier' initialization which in turn led to more successful converged solutions and a higher rate of successful convergence. This initialization works well for Sigmoid and Tanh activation functions which are centered about the y-axis, which is not the case for the ReLU activation. This can cause an issue as ReLU maps negative values to 0; thus, half of the initial gradients will be 0. He *et al.* [39] studied this problem and present an alternate initialization method for ReLU activations, commonly referred to as the 'He' initialization method.

2.2 Network Architectures

2.2.1 Semantic Segmentation

Semantic segmentation has received increased interest [43] as the applications for pixel-level understanding grows [32]. The rising number of applications for scene understanding extends from augmented reality to autonomous driving to drone inspection and surveillance. The goal of semantic segmentation is to predict the class of every pixel in the input image. In [96] the authors show that fully convolutional neural networks can accurately generate pixel-wise classification. They do this through an encoder-decoder style architecture. In this type of architecture, the image goes through several convolutional layers and is successively reduced in spatial dimensions through stride or max pooling, while simultaneously increasing in number of output channels. This portion is termed the encoder, which can be viewed as a feature extractor. The encoder network is often selected as a pre-trained network on a larger dataset such as ImageNet [20]. A

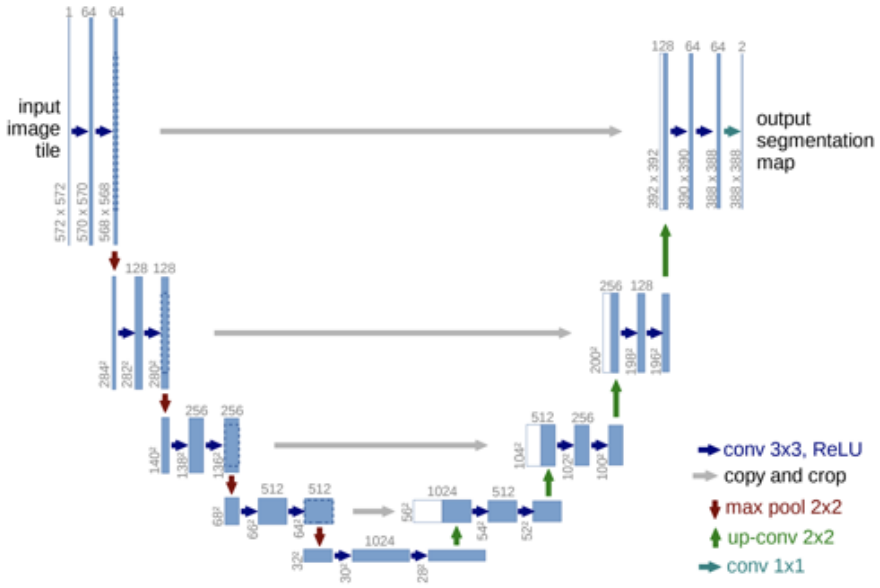


Figure 2.1: Architecture for U-Net [88].

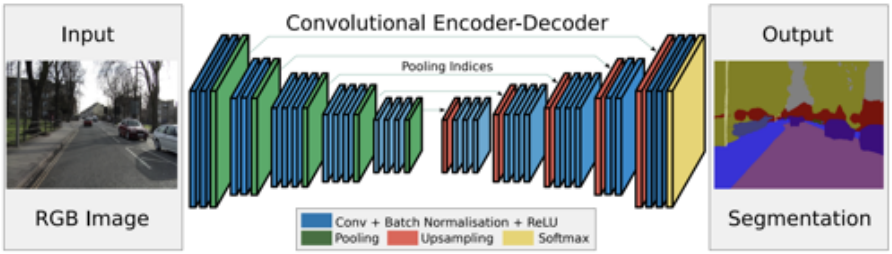


Figure 2.2: Architecture for SegNet [5].

pre-trained VGG-16 network [99] or ResNet variant [41] is often selected for the encoder. The output of the encoder is then fed into the decoder, which uses either transposed convolution or bilinear upsampling, along with additional convolution layers to upsample the features to higher spatial dimensions. The output of the decoder is then sent into a softmax layer, which produces a psuedo-probability corresponding to the prediction of the class of each pixel. Many architectures feature an encoder-decoder style architecture. Several such networks include [5, 76, 77, 88]. The architectures for U-Net and SegNet are shown in Figure 2.1 and Figure 2.2 respectively. Note that in [5] the authors report that the utilization of skip connections, in which residuals at various spatial layers in the encoder are passed directly to the corresponding spatial layer in the decoder, significantly helps the network performance. Context Encoders [81] in contrast did not use skip connections, and subsequently Pix2pix [48] independently showed that a U-Net style architecture with skip connections is able to outperform traditional encoder-decoder style architectures without skip connections. Through an ablation study, the authors showed that the skip connections enable more accurate resolution in pixel detail.

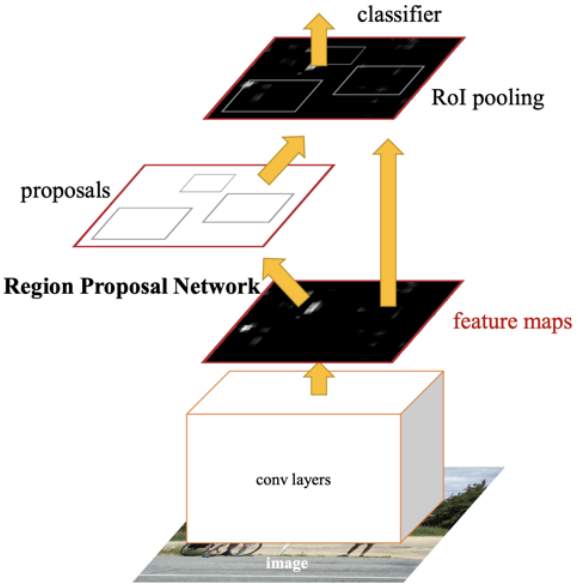


Figure 2.3: Architecture for Faster R-CNN [86].

2.2.2 Object Detection

Object detection has been heavily explored by the research community, achieving resounding success since 2014. R-CNN [36] was the first to extend the success of CNNs to object detection, and showed that CNNs could have success on other tasks, such as on the Pascal Visual Object Classes (VOC) Challenge [31]. R-CNN utilized selective search [110] to propose regions, extracted features from each region using AlexNet [58], and then used the extracted features to classify the region using a Support Vector Machine (SVM) [16].

Test time was quite slow with R-CNN (47 seconds reported in [35], since each object proposal required a forward pass through the network). SPPnets [40] and subsequently Fast R-CNN [35] drastically improved this time through the use of shared computation via spatial pyramid pooling networks. Faster R-CNN [86] finally removes the dependency on selective search, which had been the largest gating factor on speed costing reportedly 2 seconds on a CPU.

Parallel with Faster R-CNN, single shot detectors (SSD) emerged as popular detectors due to their increased speed and comparable accuracy. YOLO [84, 85] emerged as a popular single shot detector (SSD), along with Multibox [106], which was formulated as an SSD in [67]. Also in parallel, R-FCN [18] utilizes a fully convolutional network to run object detection. Object detectors and their speed and accuracy are analyzed more in depth in [46].

2.2.3 Instance Segmentation

With the success of object detection and semantic segmentation, the community started pushing for a more complete understanding of images by performing object detection with masks over the object pixels. Several early works began adding instance awareness strategies to enhance semantic segmentation networks [17, 19, 65]. In 2017 Mask R-CNN [42] surpassed prior state-

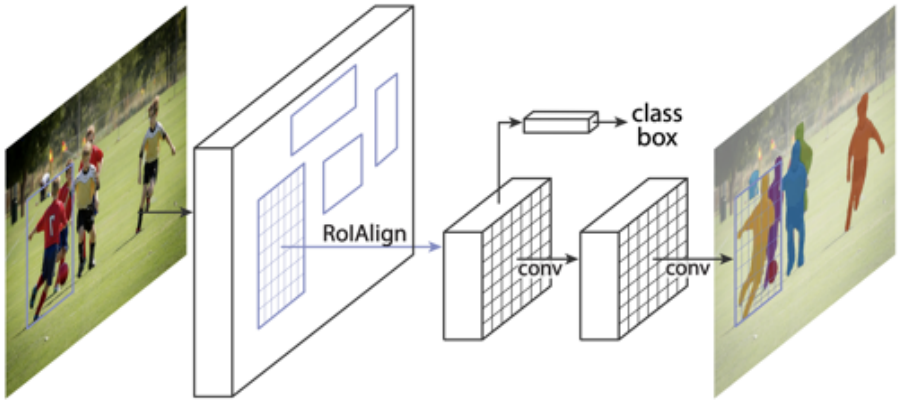


Figure 2.4: Architecture for Mask R-CNN [42].

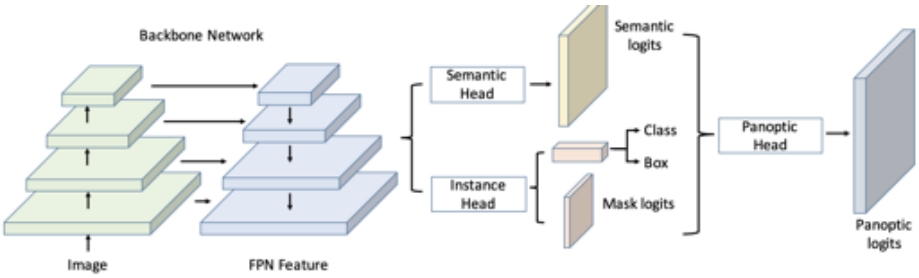


Figure 2.5: Architecture for UPSNet [119].

of-the-art methods on the Microsoft Common Objects in Context (COCO) Challenge [66] for three tracks: 1) instance segmentation, 2) bounding box object detection, and 3) person keypoint detection. The network architecture is shown in Figure 2.4. The architecture extended Faster R-CNN [86] by adding a network branch for predicting segmentation masks on each region of interest (ROI) in parallel with the branches for classification and bounding box regression.

2.2.4 Panoptic Segmentation

Panoptic Segmentation [54] is the culmination of the field moving towards a better understanding of the world from visual data. Panoptic segmentation focuses on the distinction between *things*, which you can not only classify but also detect as distinct instances, and *stuff* which you can only classify [9]. Put simply, *things* you can count, while *stuff* you cannot. Panoptic segmentation combines the task of pixel classification with instance detection. This task is relatively new, having one of the first complete systems, UPSNet [119], set to be published in CVPR '19. Their architecture is shown in Figure 2.5.

2.2.5 Datasets

Datasets have become one of the main bottlenecks in many computer vision applications. Annotating images is both expensive and time consuming. Cityscapes [15] reported 1.5 hours per each semantically segmented annotation. Looking at natural language processing, large corpuses of data have provided significant advancements in performance [38]. The ImageNet Large Scale Visual Recognition Challenge (ILSVR) dataset [20, 89] provides over 14 million annotated images for classification, and more recently focuses on detection challenges. Many networks pre-train on ImageNet to facilitate transfer learning for other vision tasks. The Pascal Visual Object Classes (VOC) Challenge dataset [31] provides thousands of annotated images for training and testing for tasks such as object classification, object detection, and object segmentation tasks. The VOC 2012 Challenge contains 20 classes, with a training and validation set of 11,530 images, and 6,929 segmentation annotations. Microsoft created the Common Objects in Context dataset [66] which contains over 200,000 annotated images with 80 object categories and 1.5 million object instances.

Since our work will focus on street-view scenes, we review several popular street-view datasets. Cityscapes [15] is one such dataset that contains roughly 5,000 images of urban street environment with semantically segmented annotations. For the task of change detection, Alcantarilla *et al.* [1] created a street-view change detection dataset containing 1,362 image pairs, accurately aligned with annotated change masks for each image pair. Mapillary [75] has released a publicly available dataset for research purposes with street-view annotated images. It contains 25,000 high resolution images with 66 object categories, and instance-specific labels for 37 classes. The KITTI Vision Benchmark Suite [34] contains a comprehensive set of vision tasks such as stereo, VO and SLAM, and 3D object detection. The 3D object detection task has 12,000 images with 40,000 objects.

2.3 Loss Functions

We focus on the use of deep networks for hazard detection and change detection. For hazard detection we utilize Faster R-CNN, and for change detection we utilize an architecture similar to SegNet. We present the loss functions used below.

2.3.1 Classification

For classification problems, crossentropy is commonly used as the loss function for deep feed-forward networks. Crossentropy is defined as

$$\mathcal{L}_{ce} = -\sum_{i} \hat{y}_i \log(y_i) \tag{2.1}$$

where $\hat{y}_i$ is the ground truth value corresponding to class i, and y_i is the predicted pseudo-probability for class i.

2.3.2 Faster R-CNN

Faster R-CNN utilizes a loss function for training the Region Proposal Network (RPN) as well as a loss function for classifying the region object into object category. Crosentropy is used for the latter, and is detailed in the subsection above. The RPN is responsible for classifying a region as object or not object, as well as regressing the coordinates of the bounding box for the region. The joint loss function for the RPN is:

$$\mathcal{L}(p_i, t_i) = \frac{1}{N_{cls}} \sum_i \mathcal{L}_{cls}(p_i, \hat{p}_i) + \lambda \sum_i \hat{p}_i \mathcal{L}_{reg}(\hat{t}_i, t_i) \tag{2.2}$$

where p_i is the predicted pseudo-probability of region i being an object, and $\hat{p}_i$ is the ground truth. Similarly, t_i are the predicted two coordinates of the bounding box, and $\hat{t}_i$ is the ground-truth bounding box coordinates. The classification loss is the same as crossentropy defined in the above section. The regression loss is defined as:

$$\mathcal{L}_{reg}(\hat{t}_i, t_i) = \sum_{i \in x,y,w,h} \mathcal{S}_{L1}(t_i - \hat{t}_i) \tag{2.3}$$

where $\mathcal{S}_{L1}$ is defined as the smooth L_1 function:

$$\mathcal{S}_{L1}(x) = \begin{cases} 0.5x^2, & \text{if } |x| \leq 1 \\ |x| - 0.5, & \text{otherwise} \end{cases} . \tag{2.4}$$

Moreover, the parametrization for the bounding box four coordinates are defined as:

$$\begin{aligned} t_x = \frac{x - x_a}{w_a} \quad t_y = \frac{y - y_a}{h_a} \\ t_w = \log(\frac{w}{w_a}) \quad t_h = \log(\frac{h}{h_a}), \end{aligned} \tag{2.5}$$

where x, y, w, and h correspond to the box's center coordinates and width and height respectively. The subscript a denotes that the parameters are of the region anchor box.

2.4 Edge Computing

The modern day computing landscape is much different than that in the 1950s and 1960s, at which point in time batch-processing mainframes were being developed. Such systems were physically large, extremely expensive, and were low compute. Then over time, the form factor of computing systems shrunk along with cost, while compute drastically increased. Today, personal computers are common, inexpensive, and provide significantly more compute. These systems are nevertheless limited in compute, however, by size and thermal constraints. For computationally intensive tasks, programmers turn to offloading computation to the cloud. This is true now more than ever thanks in large part to deep learning. The computational demand of vision algorithms have led many to turn to cloud computing. Several prominent cloud computing services are

Figure 2.6: Illustration of tiered model from [92].

Amazon Web Services [95], Microsoft Azure [2], and Google Cloud [14]. For low latency and high bandwidth applications, many have turned to edge computing [12, 29] which is able to provide higher performance due to network proximity to the client.

A pattern emerges in the computing landscape, as pointed out in [92]. The authors classify modern day hardware into a four tiers, constructing what they call a *tiered model*. They classify hardware by the dominant design constraints. Tier 1 includes cloud servers, which provide nearly unlimited compute with high power consumption. Tier 2 provides high compute with low latency due to network proximity. Tier 3 includes mobile computing, at which power consumption becomes a defining constraint. Lastly, tier 4 includes embedded sensing. An illustration of the tiered model is shown in Figure 2.6. Since our application requires significant compute, we consider tier 1 and tier 2. However, we also recognize that streaming videos over 4G LTE would saturate the network. Thus, we turn to tier 2, but make the distinction that the compute must be situated inside the vehicle. These design considerations lead us to our system architecture, which we discuss in more detail in Chapter 3.

2.5 Crowd-sourcing Applications

Road networks are immense, which makes crowd-sourcing an attractive solution. There are many applications that crowd-source road-related information. There are two tiers of crowd-sourced applications: those that are automated, and those that require a human to actively submit updates.

Google Maps [71] is one of the original and most successful road map platforms developed by Google. It crowd-sources data both automatically, as well as through active human input. It performs crowd-sourcing in an automated fashion via GPS monitoring of users that are actively using the app. This provides data in order to generate highly accurate traffic monitoring and prediction capabilities that provide users with the ability to plan the best route. Google also relies on active human efforts to keep its maps up-to-date. Google has vehicles drive the streets all over the world with 360 degree cameras to take street-view images. Furthermore, Google Maps has partner organizations that provide input data updates, as well as Google Maps users that provide local information and road map updates.

OpenStreetMap [79] is an example of such a system that requires active user input. It is a

project designed to provide accurate and public road maps. OpenStreetCam [78] is an extension of this project, which aims to improve OpenStreetMap via providing street-view images. Mapillary, in addition to having a publically available dataset for research, also crowd-sources street-view images, and has gathered millions of images. Lastly, there have been road monitoring applications in research, such as the Pothole Patrol [30], which is discussed in more detail in the related work below.

2.6 Related Work

2.6.1 LiveMap

To the best of our knowledge, no previous work has attempted to create a Waze-like real-time data collection system for road monitoring without a human in the loop. From a broader perspective, there is a substantial amount of work relating to road condition monitoring [11, 30, 74, 111]. However, none of these can handle the wide variations in hazard types and durations of events such as accidents, debris on the road, potholes, etc. Many of these systems focus on road infrastructure monitoring such as detecting potholes or general road health inspection.

A common approach for detecting potholes is to use an accelerometer with signal processing. One such example, the Pothole Patrol [30], utilizes an accelerometer and on-board filtering to determine likely locations of a pothole. Such a system has several drawbacks, the most significant being that it requires the vehicle to physically run over the pothole for detection. This is harmful to both the vehicle and uncomfortable for the driver. For this reason, it is not uncommon for drivers to swerve in order to deliberately miss running over a pothole. Furthermore, it is limited to a very specific type of road monitoring — that is, whatever the car physically hits. A non-destructive alternative is to instead use image processing.

There has also been prior research in the field of vehicular distributed network system architectures. CarTel [47] has explored the development of such a system. They detail and prototype a system that utilizes vehicles to collect sensor data, store it locally, and prioritize the dissemination of sensor data to a local server. The main disadvantage of their system is that, as described, they have no significant on-board compute and cannot locally process data-rich media, such as images. Thus, their system is limited to selective transmission as the only approach to dealing with low-bandwidth situations. Furthermore, as it relies on opportunistically connecting to local Wi-Fi hotspots, it suffers from frequent loss of network connectivity. The main advantage of LiveMap is that it can process media rich sensor data such as video feed on-board the vehicle, and can greatly save bandwidth by transmitting only distilled, interesting data, thus enabling effective use of relatively low bandwidth, but ubiquitous 4G LTE networks.

A few companies offer commercial products that are relevant to our research. Waze [115] utilizes user input in order to add detailed input to their maps. As previously discussed, the main shortcoming of such a system is that it creates dangerous situations by distracting the driver, and tends to incur high latencies and unreliable updates due to reliance on human reporting. In contrast, LiveMap is safe, automated, and real-time. Another relevant company is Roadbotics [87]. At the time of writing, Roadbotics uses cameras to capture video data of roads from the windshield, and uploads it to a server where machine learning algorithms score road conditions over

10-foot intervals. In comparison to Roadbotics, LiveMap operates in real-time and is designed to be generalizable to any hazards detectable through computer vision.

In prior work [45], it was shown that a 4G LTE network is sufficient to support such a large-scale system with tens of thousands of participating vehicles, as demonstrated through simulations in SUMO [57] (Simulation of Urban Mobility). While that work focused on the scalability challenges of using LTE, in this work we focus on the architecture, implementation, and characterization of the automated hazard reporting system.

2.6.2 Visual Change Detection

There have been prior works that explore change detection for various application domains. Stent *et al.* [102, 103] proposed a change detection system for surface inspection of tunnel linings for cracks based on images. The system involves a structure-from-motion (SfM) pipeline to create panoramas of the tunnels. Image registration is then performed on similar patches from different time instances. Finally, a two stream CNN takes as input the two image patches and performs classification.

In 2015 Alcantarilla *et al.* [1] created a full change detection pipeline, along with a novel dataset for street-view change detection. This dataset, called the Visual-Localization CMU Change Detection dataset (VL-CMU-CD), is the largest change detection dataset to date. It contains 152 scenes, each with several image pairs, totaling 1,362 image pairs total. The image pairs were extracted from the Visual-Localization CMU (VL-CMU) dataset [3, 4]. The VL-CMU dataset consists of video of the same route driven throughout various times of the year, and was originally created for long term topometric localization of a vehicle that can be invariant across seasonal changes. The geographic location is in Pittsburgh, PA, USA which exhibits significant seasonal changes that includes snow. The dataset consists of 1,362 image pairs split into a train/test split of 929 image pairs for training, and 433 image pairs for testing (taken from different sequences), with an annotated ground truth of which pixels changed along with the corresponding class. Example image pairs with ground truth are shown in Figure 4.1 in Chapter 4. The authors created the dataset using GPS and feature descriptors to select similar images and register the images, and then a computationally demaning 3D dense reconstruction pipeline to warp the viewpoint of one camera to that of the other.

More recently Varghese *et al.* [112] proposed a more modern network architecture for change detection, published in ECCV '18 workshops. It consists of a ResNet-50 encoder [41] with pre-trained weights on ImageNet, along with transposed convolutional layers to output a semantic change detection mask. However, they only trained and tested on a subset of the data, only testing on 177 out of 429 images. Unfortunately the train/test split used was unspecified, and we were unable to replicate their results with the same architecture implemented as described. Thus we cannot directly compare. There is no open-sourced code available for change detection presented in both [1, 112].

Additionally, in 2016 Suzuki *et al.* [105] proposed a semantic change detection network that utilized hypermaps for street-view panorama change detection, with stationary view point. In 2018, Lebedev *et al.* [61] trained a Generative Adversarial Network (GAN) on synthetic images of shapes for change detection, as well as a custom dataset of satellite imagery for aerial change detection.

Change detection utilizing 3D models has also been explored [80, 82]. Such 3D methods typically require a model as a prior, which is difficult to generate and error prone. In this data flow, any error in 3D model building will propagate and directly affect the performance of the change detection algorithm.

2.6.3 Visual Odometry

There are several different formulations of VO algorithms. Consequently, these algorithms can be classified as either *direct* or *indirect*, and as *dense* or *sparse*.

Indirect methods, also commonly referred to as feature-based methods, extract and match features. These features are engineered representations of high-level features in the image, usually corners. They must also store a representation of the feature that can be matched at a later frame. This introduces two large sources of error. After matching features, such methods calculate the fundamental/essential matrix or homography for planar scenes and scenes with low parallax.

Due to the high level of inaccuracies present in feature extraction and matching, such algorithms must compute the fundamental matrix or homography in a RANSAC loop. While feature-based methods have achieved accurate results, they remain computationally wasteful due to their reliance on RANSAC for robust estimation of such parameters. Several examples of such systems that use indirect methods are ORB-SLAM, ORB-SLAM2 [72, 73] and Parallel Tracking and Mapping (PTAM) [56]. Alternatively, direct methods directly use the sensor inputs, such as image intensities, to optimize an error function to determine relative camera pose.

In addition to being classified as direct or indirect, SLAM and VO algorithms can additionally be classified as dense or sparse. Dense methods have the advantage that they use all available information in the image, and can generate dense maps which is useful for robot navigation, for example. Sparse methods have the advantage that since there are less points, it is generally less computationally expensive which can lead to large computational savings, especially if the algorithm requires many iterations or a loop to converge to a solution.

There have been many iterations of direct dense methods such as direct dense VO in [101], RGB-D SLAM [51], and LSD-SLAM [26]. Even using dense methods, these systems achieve real-time performance on modern CPUs due to the highly efficient nature of these types of algorithms. More recent advances highlight the fact that the information contained in image intensities are highly redundant, and attempt to minimize the photometric error only over sparse random points in the image in order to increase efficiency and thus speed [27]. Another direct method that has been used with success is the iterative closest point (ICP) algorithm, which is used in systems such as [49, 116]. These systems minimize the difference between point alignment in contrast to image intensities.

The extension of direct methods using edge pixels is a logical direction, yet to the best of our knowledge no work has solely used edge pixels in a direct method minimizing the photometric error. In [93] the authors reduce a Euclidean geometric error using the distance transform on edges which does not utilize all information available in the scene. In [118] the authors minimize a joint error function combining photometric error over all pixels along with geometric error over edge pixels. Minimizing a joint error function always suffers from the decision on how best to weight each function, and the weighting can have significant effect on the final converged

solution. In [27], the authors threshold by gradients, which does not guarantee edges due to noise. They additionally select texture-less regions as well.

We hypothesize direct methods have often avoided solely using edges due to several pitfalls when extending direct methods. In particular, there is the question of which edges to use among the reference and the new image. We have found that selecting the wrong edges produces incorrect convergence. Additionally, edges are inherently unstable as changes in intensity and geometric position result in large changes in intensity and geometric position respectively, and incorrect formulation of the problem results in an incorrect solution. There have also been several indirect systems that have experimented with various strategies for utilizing edge information to track camera pose [23, 55, 70, 107]. Such methods treat edges as features and use complicated matching strategies which increase computation and add unnecessary heuristics. In contrast our method simply extracts edges and incorporates it in a direct method. This simple yet elegant and highly efficient sparse direct method provides lower drift than previous state-of-the-art visual odometry methods.

Any system that extracts edges must choose between several edge extraction algorithms. The most prominent are Canny edges [10], followed by edges extracted from Laplacian of Gaussian (LoG) filters which are efficiently implemented using Difference of Gaussians (DoG). Another type of edge that is not as popular but is very simple are Sobel edges. More recently, there has been research involving the learning of edge features. In [21] the authors utilize structured forests, and in [117] the authors utilize deep learning. Instead of selecting one, we test various edge extraction algorithms with our system select the optimal edge extraction algorithm. Note that [117] requires the use of a GPU and is far from real-time, so we do not consider this method.

Chapter 3

LiveMap

3.1 System Architecture

LiveMap [13] is designed to scale to a large number of vehicles [45], and it is critical that our system scales with increased bandwidth such that we do not saturate the 4G LTE network. Sending video streams from every vehicle would quickly prove to be intractable. Instead, we harness compute ability both in the vehicle and in the infrastructure on the edge to reduce network usage.

The key idea behind LiveMap is that it performs the heavy video analytics computations on-board the vehicle. Each participating vehicle is required to have an on-board camera and GPS module, as well as a computer, called the Vehicle Cloudlet. As the vehicle drives around, it runs a Convolutional Neural Network (CNN) to detect various hazards. When a hazard is detected with a confidence level greater than a threshold, the Vehicle Cloudlet reports it to the Zone Cloudlet via a message containing the GPS coordinates, the annotated image identifying the hazard(s), as well as the timestamp and other metadata pertaining to the drive. The Zone Cloudlet is responsible for handling the incoming data, storing it in a database, and notifying the vehicles in the vicinity of the hazard. See Figure 3.1 for a high-level overview of LiveMap. The Zone Cloudlet is situated on the edge in order to provide more localized control of user data and privacy, as well as for potential national security reasons to decentralize such information [91]. This is in contrast to Vehicle-to-Vehicle (V2V) Communication systems [97, 120], which focus on addressing issues such as immediate collision detection and avoidance, as well as highway platooning. V2V communication exhibits additional security concerns, such as message accuracy and reliability, etc. The Zone Cloudlet in contrast can address such issues by vetting information before sharing it with other vehicles. Note that this pipeline would also enable autonomous vehicles to make reports as well since it does not require a human in the loop.

There are several challenges associated with choosing when and how often to report hazards. One such challenge is avoiding identical reports. For example, a vehicle stuck in traffic behind an accident will repeatedly detect the same accident until the accident clears. Furthermore, many other vehicles in-sight of the accident will also detect this accident. This single event may be reported thousands of times. Such careless report sending policies would quickly saturate the network with useless duplicate data.

To address this issue of duplicate data, we employ a policy that if a vehicle detects a hazard

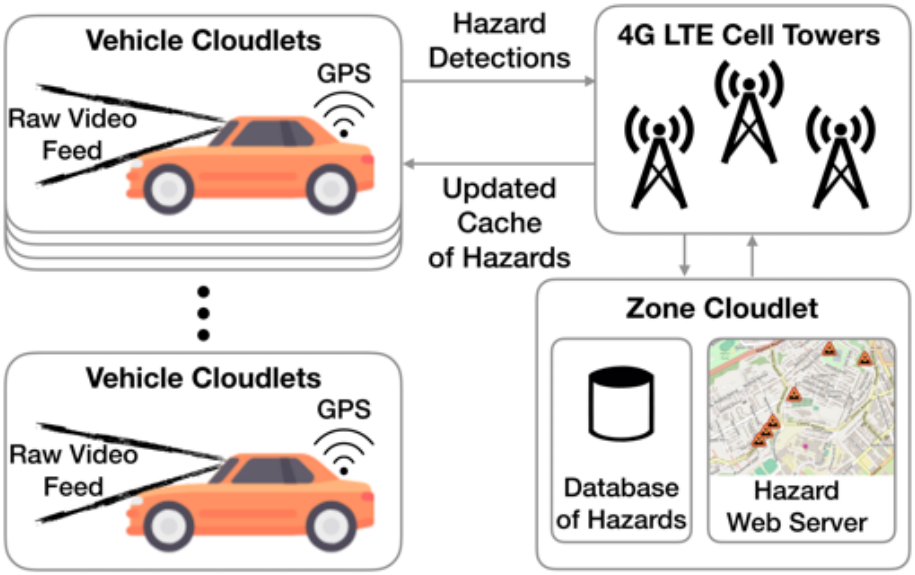


Figure 3.1: System Overview. Vehicle Cloudlets run hazard detection on live video feed, reporting any hazards to the Zone Cloudlet for dissemination to other vehicles and for display through a web interface.

of the same type in close proximity to one that has previously been reported, it will not send the report. This achieves the desired result of saving bandwidth by avoiding duplicate reports, with the small cost of perhaps missing a few instances of the same class. We argue that this is indeed an acceptable tradeoff with the following example: several objects fall from a truck in one area on the highway, creating hazardous debris. A car driving by will detect multiple frames of debris, but will only send one report. In general, this single report for multiple related hazards should suffice.

Another such issue that complicates this design decision is the issue of hazard removal. Different types of hazards may have various temporal lifespans. For instance, some hazards like car accidents are usually cleaned up and removed within a matter of hours, while potholes can exist for several months. At the Zone Cloudlet, detecting when a hazard is no longer present can be difficult. A naive solution would be to wait until reports of the hazard stop arriving. This idea is flawed due to our previous design decision to not report duplicate hazards. It is difficult to know if a hazard has "expired" or if the duplicate elimination policy is preventing further reports. It is possible the Zone Cloudlet may never hear again about a long-lasting hazard.

To address this second issue, we use a polling scheme. In this polling scheme, the Zone Cloudlet occasionally sends a message to vehicles near a previously-reported hazard asking them if they still see the hazard, and optionally whether or not to send an image. The Vehicle Cloudlets then send a "yes" or "no" reply back after validating the continued presence of the hazard, minimizing the bandwidth consumed for verifying the hazard presence. The rate at which such verification polls are sent would be inversely proportional to the expected duration of the hazard, based on the mean or median duration of the hazard class type. This can be further optimized as the system collects more data and can generate more accurate predictions of when best to poll

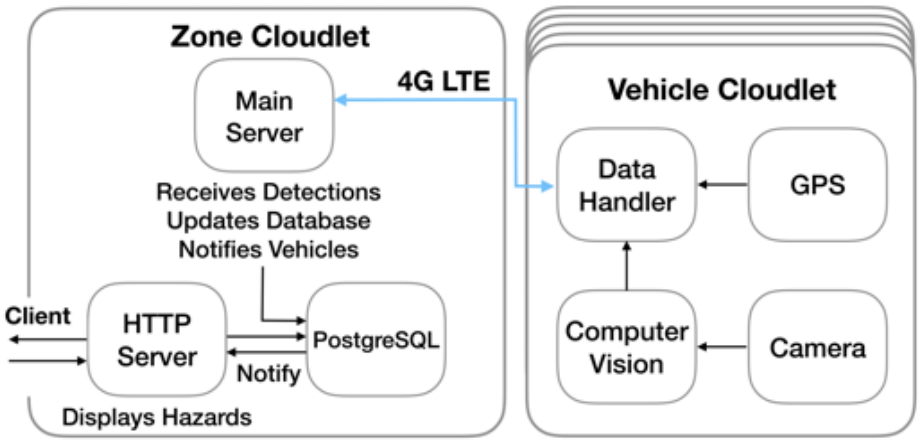


Figure 3.2: System Architecture. The Zone Cloudlet handles incoming hazard reports and notifies other vehicles. It also runs an HTTP server that displays a map of detected hazards.

while minimizing network usage.

3.2 Implementation

3.2.1 Zone Cloudlet

In our experiments, the Zone Cloudlet is situated on the edge and is implemented on a server on the Carnegie Mellon University (CMU) campus. Note that for cost reasons it might make more sense to host this entity in the cloud. For security or privacy reasons it could also make more sense to keep it at the edge. The Zone Cloudlet has a number of responsibilities, which can be classified into two main functionalities: 1) hazard message operations and 2) web server operations. Within the class of hazard message operations, the Zone Cloudlet must accept and handle incoming hazard reports from vehicles, and transmit update data to vehicles. See Figure 3.2.

When the Zone Cloudlet receives a hazard report, it queries its database for any matching hazards previously reported within the tolerance of the GPS module (typically around 3-5 meters). If no active matches are found (i.e., this is not a duplicate report), it adds the hazard to its database. It then sends a message to all vehicles in its area of responsibility with the GPS coordinates of the hazard, and an annotated image identifying the hazard.

For our implementation, we opted to use the Message Queuing Telemetry Transport (MQTT) protocol, an ISO standard built on top of TCP/IP [24]. MQTT is a publisher-subscriber-based messaging protocol intended for the "Internet of Things," and is designed around the idea that machine-to-machine communication will have limited network bandwidth and will suffer from intermittent connectivity. This fits our application requirements, since moving vehicles will invariably be in a dead zone at one time or another, and the 4G LTE network has limited bandwidth. For a Vehicle Cloudlet to receive updates from the Zone Cloudlet, it simply has to look up its GPS coordinates, find the nearest Zone Cloudlet, and subscribe to the updates being published

by the Zone Cloudlet. In order to avoid major security concerns, we limit communication in LiveMap as follows: the Zone Cloudlet is the message broker, and the Vehicle Cloudlets are the clients; no vehicle-to-vehicle communication is performed, and all messages to the Vehicle Cloudlets must come from the Zone Cloudlet.

The second class of operations has to do with the presentation of hazard information, which is done via a web-based interface. The Zone Cloudlet doubles as a web server, and actively delivers hazard information on a map overlay to connected web clients. When a client, say a city official, connects to the web server, the Zone Cloudlet sends all present hazards to the client, which are displayed on the map as icons. The user can then click on the icon and have additional information displayed, such as GPS coordinates and the annotated image of the detected hazard. When a new hazard is added to the database, a notification is sent via web sockets to all connected clients in real-time. An example screenshot of the web-based hazard display is shown in Figure 3.3. We use Leaflet [60] as our map serving framework, and Node.js to dynamically deliver content. This web server display can effectively serve as a quality control measure. Since the details of all hazards can be displayed on the map as annotated images, a city official can easily verify the accuracy of hazard detections with a click on each hazard icon. This provides an interface for human oversight of the system, letting an official reject any false positives before notifying the appropriate response teams, for example.

3.2.2 Vehicle Cloudlet

The Vehicle Cloudlet performs image processing to find hazards. It utilizes a CNN to perform object detection to identify road abnormalities, and then sends a message to the Zone Cloudlet with accompanying data. When it detects a hazard, it checks its local database of current hazards for a nearby hazard of the same type. If it doesn't find any, it adds it to its database and sends a message to the Zone Cloudlet. In doing this it avoids repeatedly sending notifications of known hazards. When the Vehicle Cloudlet receives a hazard notification from the Zone Cloudlet, it adds it to its database.

The precision and recall capability of the sensing is a function of compute capability, which is a function of cost. We explore the trade-off between precision/recall and cost by experimenting with two different designs and implementations for the Vehicle Cloudlet. One configuration is a powerful server with state-of-the-art compute capability but can run reliably off of a car alternator. We call this the Big Vehicle Cloudlet (BVC). It can afford to use a more computationally expensive and memory intensive CNN architecture for detection, employing dual GPUs with high bandwidth and large memory. The second option uses a mobile phone as the Vehicle Cloudlet, which has significantly less compute capability and memory, but is an order of magnitude lower in cost. We term this the Small Vehicle Cloudlet (SVC). We outline both implementations below and highlight the key differences between them.

3.2.3 Big Vehicle Cloudlet

The first system we test is a ruggedized server, configured with 2 Intel® Xeon® Processors, 2 Nvidia Tesla V100 GPUs, and a liquid cooling system, shown in Figure 3.4. This system configuration can afford to run a large CNN model with a large number of weights, which is

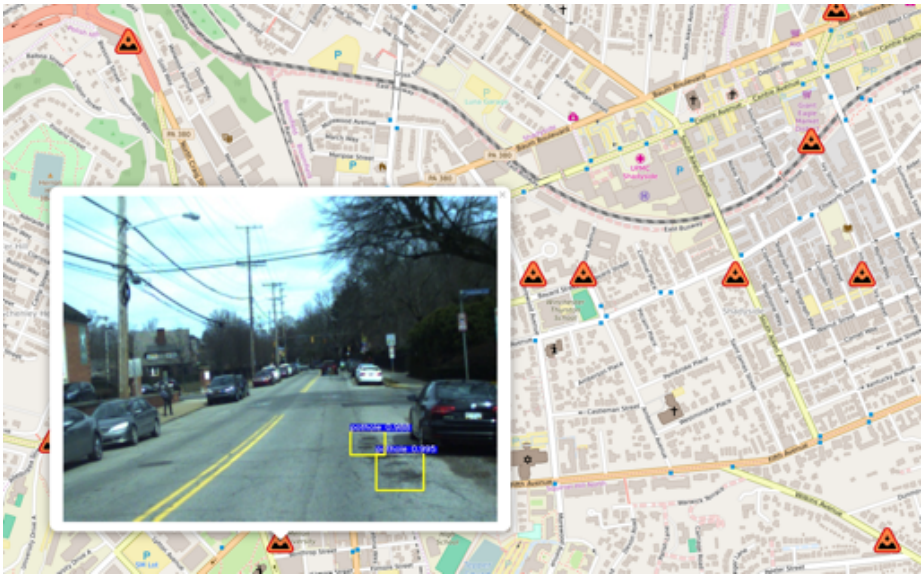


Figure 3.3: Map of Hazards. Icons for each hazard appear real-time. Clicking on each icon displays more hazard information.

Figure 3.4: Big Vehicle Cloudlet setup in Navlab. The top unit is the pump with coolant, and the bottom unit is the cloudlet.

both memory and compute intensive. On this system, we run Faster R-CNN [86] as our object detector, which provides state-of-the-art accuracy, but is computationally demanding. The image processing is run on both GPUs independently in order to double the processing frame rate. Each GPU has a copy of the CNN weights and can run inference on an individual image independently. The output image is the original image overlaid with annotations indicating where a hazard was detected. If a hazard is detected and is not a duplicate, the Vehicle Cloudlet prepares a message and sends it to the Zone Cloudlet. If the image is not interesting and no hazards were detected, the system discards the image.

This setup provides the best scalability, as we have moved all of the compute to the vehicle, and the aggregate compute capability will scale with number of vehicles. Furthermore, the BVC is well-positioned to address privacy concerns that arise from recording people in such video feeds. The BVC has enough compute to denature images, or remove privacy sensitive information such as blurring faces, as done in [98, 113].

Table 3.1: Latency Measurements in ms

Config	Detection	RTT	End-to-end
BVC	38.6 (3.1)	205.6 (50.2)	244.2 (50.3)
SVC+ZC	391.6 (67.1)		597.2 (83.7)

BVC: Big Vehicle Cloudlet, SVC: Small Vehicle Cloudlet
ZC: Zone Cloudlet, std. deviation in parentheses
Note that the latency for BVC is per GPU

3.2.4 Small Vehicle Cloudlet

The second system we test uses a smartphone-class device as the Vehicle Cloudlet. As this platform is not capable of running a large CNN used for hazard detection, we employ the early discard method proposed by [114] that uses lightweight computations to selectively send only the interesting images to the Zone Cloudlet, which would then run the hazard detection algorithm. The small vehicle cloudlet is limited to making send-don't send decisions using a small and simple neural network model. The expensive hazard detection algorithm is then run on the Zone Cloudlet. This significantly reduces costs of the vehicles, but comes at the expense of scalability, since we move the hazard detection to the centralized Zone Cloudlet.

We implement the Small Vehicle Cloudlet using a Nexus 6 smartphone, and run MobileNet [44] as the image classifier. This has significantly lower computational requirements than Faster-RCNN, and can process roughly three frames per second on this platform.

3.3 Experimental Results

There are three aspects of performance that we consider when evaluating LiveMap. The first is the end-to-end system latency for LiveMap given a new hazard. This is the time it takes from the point at which a vehicle detects a hazard, sends it to the Zone Cloudlet, and the Zone Cloudlet sends out the newly captured data to nearby vehicles. The second evaluation criterion is the hazard detection precision and recall. Ideally we want high precision (quality detections with a high ratio of true positives among all detections), as well as high recall (most hazards are actually reported). Lastly, we need to quantify the bandwidth savings by moving compute into the vehicle.

System latency can be further broken down into two categories: 1) detection latency, or the time it takes to process a single image, and 2) message round-trip latency, or the time it takes to send a message to the Zone Cloudlet and receive an acknowledgement. The detection latency is dependent on the Vehicle Cloudlet server, while the round-trip time (RTT) is the same for both configurations since they will both be using 4G LTE for transmission. Note that for the Small Vehicle Cloudlet, the detection latency includes both the local image classification, which gives a send-don't send result, as well as the actual hazard detection run by the Zone Cloudlet to annotate the hazards. To test the detection latency, we record the time the system takes to process each frame over a one minute interval. See Table 3.1 for RTT and Detection Latency results. Note that the Big Vehicle Cloudlet has two GPUs, but Table 3.1 shows latency *per* GPU.

Figure 3.5: Examples of annotated pothole detections.

On an end-to-end basis, using the Big Vehicle Cloudlet incurs less than half of the latency as using the Small Vehicle Cloudlet.

The camera frame rate is 30 fps, therefore the ideal processing latency is below 33 ms to achieve real-time performance. Per-frame processing times greater than 33 ms would imply that frames are dropped, or not processed. By utilizing 2 GPUs and alternating frames assigned to each, the Big Vehicle Cloudlet can avoid dropping any frames. While it is great to process every frame, it is often not necessary in order to detect hazards.

Our second metric attempts to quantify how well the system actually detects road hazards. We consider this in two different ways. For the Big Vehicle Cloudlet we measure the mean Average Precision (mAP), which is a common metric for evaluating bounding-box-based object detection algorithms. We use the standard mAP_{50} which defines a correct detection if the intersection over union of the detected and the ground truth bounding boxes is greater than 50%. Note that we use this metric to evaluate BVC's detection algorithm, which is also run on the Zone Cloudlet for the SVC configuration. We obtain this metric over all test images.

Frequently a hazard will be encountered more than in just one frame. In fact, we expect to encounter any given hazard in multiple frames. Even if detection failed in one frame, the system may still be able to identify the hazard in another one. To address this, we employ an event-level recall metric, which we define as the number of distinct hazards correctly identified over the total number of hazards. For the Big Vehicle Cloudlet we filter out duplicate detections based on GPS location. If we detect a hazard of the same class in the same location, we can filter out the message as it was likely the same instance of the hazard previously detected. Note that we cannot utilize this for the Small Vehicle Cloudlet, as it only categorizes the image as interesting

Table 3.2: Detection Results

Config	FPS	mAP	Event Recall	Avg. Mbps
BVC	30	52.7	92.3%	0.46
SVC+ZC	2.8	34.3*	84.6%	0.91

BVC: Big Vehicle Cloudlet, SVC: Small Vehicle Cloudlet,
ZC: Zone Cloudlet, *MobileNet at 0.9 Recall

or not interesting, and furthermore runs at a much slower frame rate.

Deep Neural Networks require large amounts of annotated data for training. For our prototype, it was not feasible to collect a large training set of accidents or debris on the road. Rather, we focused our proof-of-concept on detecting hazards for which we could collect data, namely potholes. We annotated approximately 3,000 pothole images to train our detector from a set of set of driving videos we collected. Data collected consists of footage in the greater Pittsburgh region across various lighting and weather conditions, as well as from various viewpoints. We kept aside a portion of the data for testing. We show the metrics in Table 3.2 and example positive detections in Figure 3.5. Due to the limited processing rate of the SVC configuration, many frames are dropped. This reduces its consumed bandwidth, but also reduces its event recall.

Finally, the third metric is the average bandwidth saved by utilizing compute in-vehicle. We run both the Big and Small Vehicle Cloudlets on the same recorded driving video, and record the amount of data transferred over TCP. We compare these to a third baseline option, in which the compressed H.264 video is streamed to a central server for processing. Figure 3.6 summarizes our results. Here, the video stream rate of the baseline is plotted as a red line. The GPS-based duplicate suppression is heavily dependent on the speed of the vehicle, and the rate at which we encounter hazards. Therefore, we test the Big Vehicle Cloudlet with several different radii parameters for redundant hazard checking, ranging from 1 meter to 25 meters. We show the theoretical best detector as the green star in the bottom right for reference, which exhibits perfect precision and recall, as well as the lowest possible bandwidth (i.e., each unique hazard is reported exactly once). Note that the consumed bandwidth is inversely proportional to the detection precision.

We can see that the Big Vehicle Cloudlet performs the best in terms of recall with GPS filtering of radii 1m, 3m, and 5m. Using a larger GPS filtering radius decreases bandwidth consumed at the cost of hazard recall. The Small Vehicle Cloudlet provides a reasonable compromise between the two. The SVC requires more bandwidth since it uses a ”filtering out” approach, and sends the image without knowing what the predicted class type is due to its limited memory and compute capabilities. We cannot use such spatial filter strategies because the SVC does not know what class of hazard was just detected. This is the main drawback of such a filtering out approach, and it therefore tends to send images more frequently on average. Additionally, since it operates at a low frame rate, it may be susceptible to missing hazards that are only visible briefly, a situation not reflected by these experiments. Overall, moving compute inside the vehicle reduces the average bandwidth consumed by around 95% compared to the baseline. Streaming the data costs nearly 9 Mbps uplink while our system used less than 0.5 Mbps with 5 meter GPS filtering. Extrapolating from this, if the average vehicle is driven for 1 hour a day, over the course of a month streaming video would require approximately 121 GB per vehicle, whereas our method

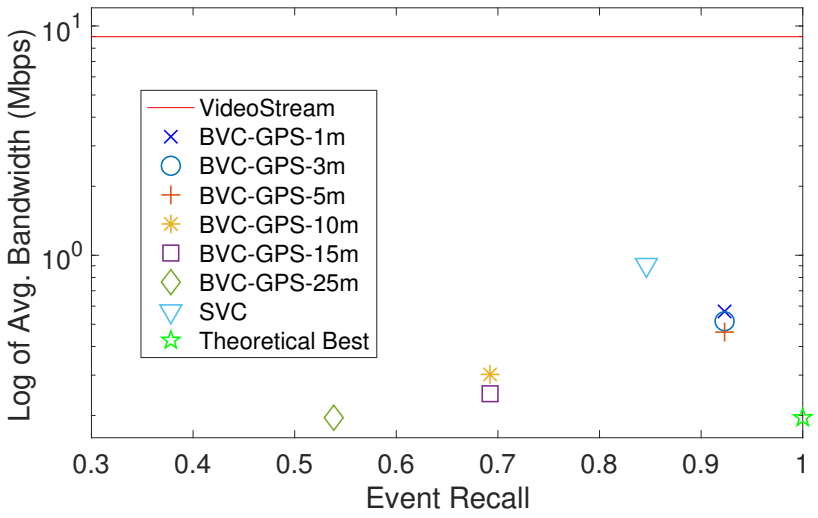


Figure 3.6: Bandwidth uplink in Mbps. Y-axis shown in log scale.

would consume only 6.3 GB.

In Figure 3.6 we can see that even the theoretical "perfect detector," which exhibits perfect precision and recall, still sends a significant amount of data detecting potholes. Further restricting the report rate for non-urgent hazards would further reduce this number as well, as potholes in some cities may be categorized as non-urgent and can be transmitted when connected to WI-FI. This result illustrates the need to treat different types of hazards with a varying degree of importance. One such solution is to for each Vehicle Cloudlet to contain a list of hazard types with a possibly dynamic importance ranking for each hazard type. Hazards that are ranked with high importance would be sent immediately, while those that are less serious can be deferred until the Vehicle Cloudlet is connected to WI-FI in a garage, for example. This feature would save bandwidth without sacrificing completeness.

3.3.1 Additional Hazards

Prior work [45] has shown the detection of deer on the roadside[1]. We have also recently demonstrated the system detecting traffic cones[2], which often signal lane closures. See Figure 3.7, 3.8, 3.9, and 3.10 for examples of LiveMap detecting traffic cones. Note that for these live demonstrations we include minor adjustments such as relaxing the constraint on hazard filtering based on GPS coordinates, and we continuously update the Zone Cloudlet with the vehicle's position (shown as a red vehicle, with previous positions marked with a blue dot). The list of objects that can be detected is extensible: new object classes can be added to the system by providing a classifier trained on the object data. Some other items that may be useful to detect include road closure signs, road debris, construction equipment, and accidents. New detectors can be integrated as they become available.

[1] https://youtu.be/_GrP42359z8
[2] https://youtu.be/TToOb2rTNZU

Figure 3.7: LiveMap example live run.

Figure 3.8: LiveMap live run detection example.

Figure 3.9: LiveMap live run multiple detection example.

Figure 3.10: LiveMap live run multiple detection with missing detection example.

Chapter 4

Visual Change Detection

In the previous chapter, a prototype for LiveMap was developed and validated. One limitation of the system as presented was the inability to visually detect if a hazard had disappeared. If a traffic accident or road closure sign was reported, a polling scheme was required to determine if a hazard had disappeared. In order to overcome this limitation, visual semantic change detection is required. More generally, semantic change detection provides the capability to monitor not only road hazard updates, but also changes corresponding to various map aspects outside those pertaining to hazards. For example, perhaps we wish to determine if a store has went out of business, a road sign has been damaged or removed, or if there is graffiti on the side of a building. In this chapter, we present a pipeline for detecting change via image registration followed by a deep CNN. Note that this framework is generalizable to any class of change that we wish to detect. In principle we can train it on the changes we wish to detect such as signs, bins, etc., and it will learn to ignore changes that we do not care about, such as seasonal changes or weather.

4.1 Motivation

Visual change detection is a task that has many potential use cases. Such use cases include crowd-sourcing map updates for regular or autonomous vehicles, smart cities that wish to react quickly to road closures or accidents, surveillance of property or regions of interest for both private and military industries, or even geographic surveying. Change is a certainty, and with decreasing cost of camera sensors coupled with ever-increasing ubiquity, this task presents an equally growing opportunity to benefit the community.

Change detection has a value proposition across the entire spectrum of transportation modalities. Drones can utilize change detection to survey large areas from an aerial perspective. Yet surveillance with drones is just one use case on one platform. Vehicles can utilize change detection to aggregate high-level and high-detail map updates for faster accident responses, and smarter and safer route-planning. Wheeled robots can utilize change detection in tunnels, pipes, and other hard to reach places for humans for inspection purposes. Aquatic robots can utilize change detection to inspect the hulls of ships, a costly yet important endeavor. Medical imaging can utilize change detection to detect anomalies and identify health problems for individuals. Even stationary surveillance systems that may utilize low-level background subtraction would

Figure 4.1: Example Images from VL-CMU-CD [1], along with ground truth classes and corresponding colors.

see improvements with the added semantic detection capability and more robust information provided by change detection.

Visual change detection is a difficult problem due to the fact that between any two images being compared, there will not only be photometric differences due to changes in illumination, contrast, sensor noise, resolution, and image alignment, but also differences due to pose differences of deformable objects or occlusion. Moreover, the definition of change varies across applications. There are many scenarios where change detection would be highly beneficial, however the definition of what constitutes change varies on a case to case basis. There will invariably be differences between images that we wish to detect in certain scenarios, and wish to neglect in others. This is commonly referred to as *changes of interest* vs. *nuisance changes* [1, 83]. This highlights a key observation- for change detection we need a flexible solution. CNNs are uniquely poised to address this problem due to the fact that they can be trained in a supervised fashion to detect change. This means that in each case, the user can define exactly what they wish to be detected. CNNs are therefore an excellent solution for such a problem.

Although humans are very good at detecting changes across seasons and long periods of time, it is not feasible for majority of applications to have a human in the loop. Image streams present an interesting challenge where there is a large amount of data at high bandwidth, and yet human attention is a precious and limited resource. Automating visual tasks has exploded within the vision community, yet change detection has somewhat lacked in popularity within the community despite growing potential for such an system.

4.2 Problem Statement

We restrict our domain to street-view change detection in order to aid in the crowd-sourcing of accurate, and live map updates. We propose the following scenario: a vehicle drives down a route, and uploads images and GPS information for offline processing. A second vehicle drives down the same route at a later point in time. Utilizing the data collected and any processing done offline from the first vehicle, we wish to detect at a pixel level what has changed along the route. We use various sequences from the VL-CMU dataset to test our system.

4.3 System Pipeline

In order to solve this problem, we require a way to match an image of the current trajectory to the closest visually matching image of the previous trajectory. An overview of our proposed system is shown in Figure 4.2. We select the best matching image in a two-step process. We use GPS coordinates as a coarse selection for matches, and then use feature matching as a fine selection to find the best match. When down-selecting images using GPS coordinates, we query a database of the latitude and longitude of previous images based on the GPS of the current image and find the 15 closest images. Next we match SIFT [68] features on those 15 closest images to find the best image match, and warp the image to the view-point of the current image by finding the corresponding homography within a RANSAC loop. An alternate solution would be to create a dense 3D reconstruction of the trajectory similar to [1] and perform the 5-point algorithm for more accurate view-point alignment. However, we found that our method works well for two reasons. The first is that the GPS coordinates are fairly precise across various runs in the VL-CMU dataset. The second is that the view-point changes were small enough that we could use the planar world assumption and avoid additional computational complexity from more computationally demanding alternatives such as estimating the Essential or Fundamental matrices. Now that we have two approximately aligned images at two different time instances we can feed the pair into our change detection CNN, which is described in more detail in the following section. The output of our network is a semantically labeled mask corresponding to the pixel-level change in the image pair.

4.4 Network Architecture

The full network, detailed in Figure 4.3, consists of a common encoder which encodes both images, followed by two decoders. The output of the two decoded images are concatenated

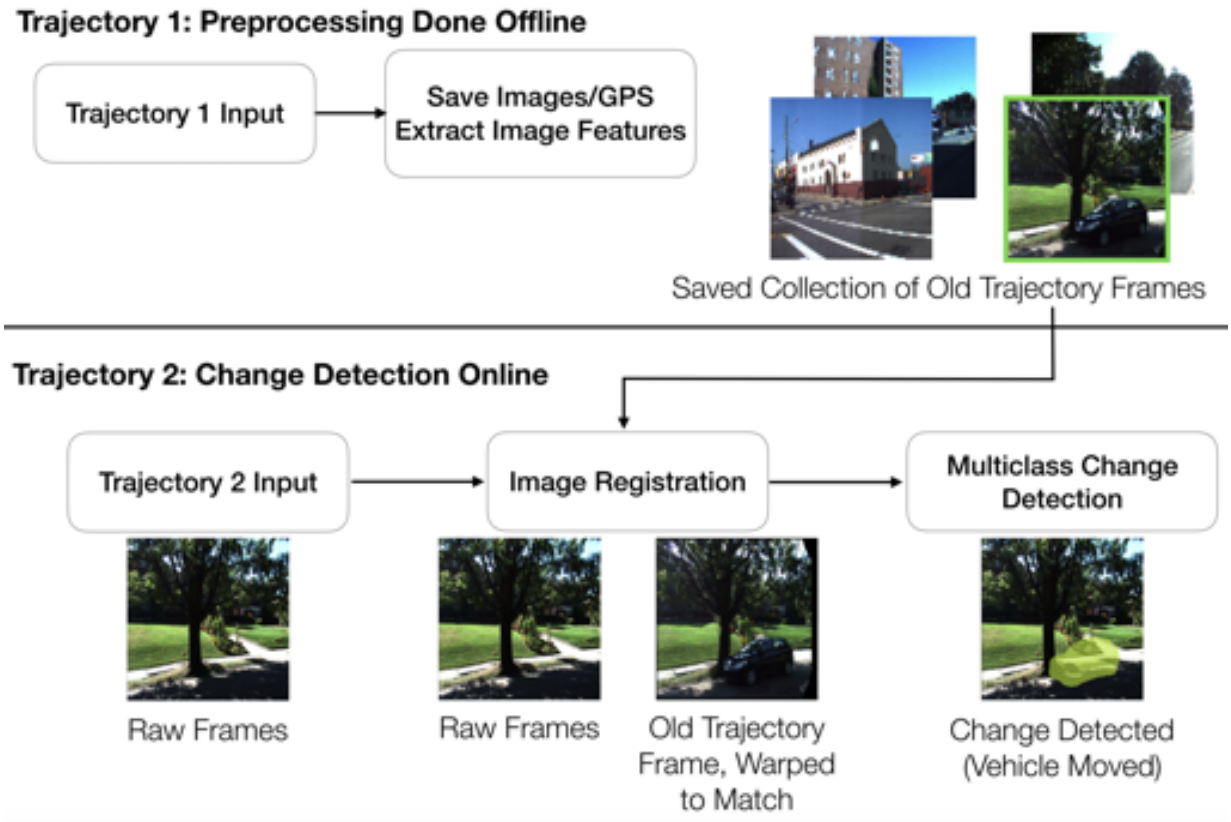


Figure 4.2: Change Detection full system pipeline.

channel-wise, and sent through three final convolutional layers before a softmax output. We use ReLU for all activation functions aside from the final softmax. The final output of our system is a tensor corresponding to $h \times w \times$ # classes. Motivated by a U-Net style architecture [88] which has been shown to work well on segmentation tasks, we create a U-Net encoder-decoder with skip connections. We capitalize on transfer learning by using VGG-16 pre-trained weights on ImageNet to initialize the first 16 layers of our encoder. We add 3 convolutional layers after the VGG-16 layers to the encoder. The decoder is symmetric to VGG-16 with the caveat that we re-use the same block structure, always using one transposed convolution followed by three convolutions. We independently came to a similar architecture to SegNet [5], with several important functional differences. We perform skip connections prior to the max pooling output to preserve information of finer details, and add convolutions between the encoder-decoder to avoid a sudden jump from max-pooling to an upsampling layer which seems unintuitive and wasteful.

4.4.1 Training

We train on the VL-CMU-CD dataset which contrains 933 image pairs, $T1$ and $T2$, in the training set. We use the Adam optimizer [53] with a learning rate of $1e^{-4}$. For data augmentation, we use affine transformations, along with image switching for $T1$ and $T2$ images. We also occasionally

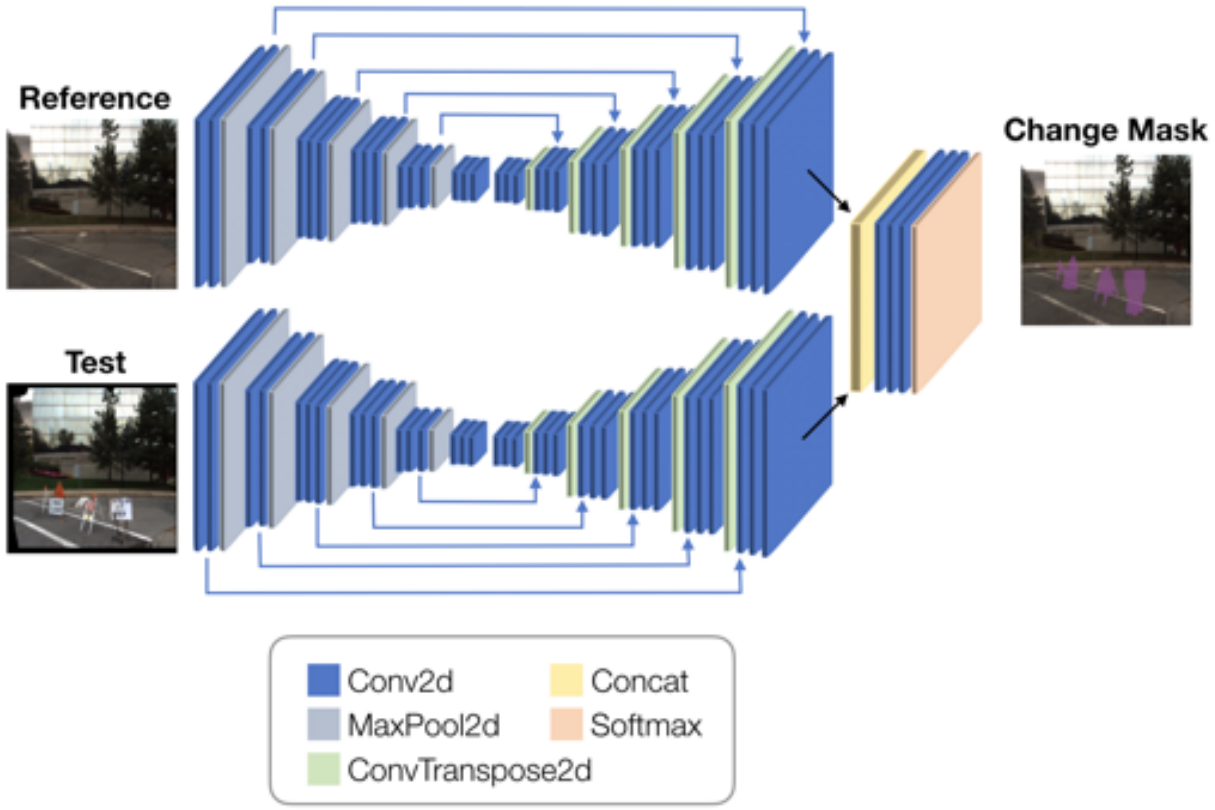


Figure 4.3: Architecture for Change Detection. Note that the encoder weights are tied, however the decoder weights are learned separately.

use the same image for both $T1$ and $T2$ images, in which case we use *No Change* for the ground truth.

As provided, there are 11 classes in the dataset. However from inspection of the dataset, there are two conflicting classes- *Miscellaneous* and *Other Objects*. There are only a couple of images of *Miscellaneous* in the training set, and zero occurrences in the test set. Furthermore, the objects labeled *Miscellaneous* in the training set clearly belong to the *Rubbish* class label. We therefore make this change, thus our output corresponds to 10 classes, as shown in Figure 4.1.

4.5 Experiments

We benchmark our system against prior work on the VL-CMU-CD dataset. We compare common classification metrics such as *Precision* and *Recall*. We convert our multi-class predictions to binary output in order to plot the ROC curve and Precision-Recall curve for direct comparison to prior works. Our results are shown in Table 4.1 and in Table 4.2. We define binary pixel accuracy as the fraction of pixels correctly classified for the entire image. Since the class *No Change* dominates the dataset, we recognize that this is a poor metric to understand network performance by itself, which is why we include the binary Precision-Recall curve and ROC curve which are

Table 4.1: Quantitative Comparison for our method compared to others for binary change detection on the VL-CMU-CD dataset.

	FPR=0.10			FPR=0.01		
Method	Pr	Re	F_1	Pr	Re	F_1
D-SIFT [68]	0.20	0.31	0.24	0.25	0.04	0.07
DAISY [109]	0.15	0.22	0.18	0.11	0.02	0.03
DASC [52]	0.20	0.30	0.23	0.29	0.05	0.08
CNN [90]	0.31	0.57	0.40	0.48	0.12	0.18
CD-Net [1]	**0.40**	0.85	**0.55**	0.79	0.46	0.58
ours	0.38	**0.87**	0.53	**0.82**	**0.63**	**0.71**

Table 4.2: Quantitative Evaluation when taking argmax of softmax output.

Method	Binary Acc.	Multi-class Acc.	Binary Pr	Binary Re	Binary F_1
ours	96.23	66.92	59.16	80.60	68.22

displayed in Figure 4.4 and Figure 4.5 respectively. We define multi-class pixel accuracy as the fraction of pixels correctly classified over the pixels correctly identified as change.

4.6 Discussion

As can be seen, our system performs quite well on the task of binary change detection. We compare to prior methods, including D-SIFT [68], DAISY [109], and DASC [52] which utilize dense descriptors to predict change, as well as the CNN used in [90] and CD-Net [1]. We observe that the methods utilizing CNNs perform significantly better than their dense descriptor counterparts. Furthermore, our network performs significantly better compared to prior methods, especially at lower false positive rates. Note that the VL-CMU-CD dataset is imbalanced, with majority of the pixels labeled as *No Change*. For this reason, the Precision-Recall curve is best for comparing existing methods.

Moreover, our network does quite well on the task of semantic change detection as shown in Figure 4.7 and Figure 4.8. Our system does very well detecting classes such as bins, construction, vehicles, rubbish, signs, and even other objects. Classes in which it tends to miss are traffic cones, person/cycle, and scenes where the construction corresponds to fences. We additionally plot the Precision-Recall curves for each class in Figure 4.6, using a one vs. rest approach. We examine the test image output, and discuss limitations of our network along with limitations of the dataset below.

4.6.1 Failure Modes

Our network has two main failure modes: 1) failing to detect change, and 2) mis-classifying the change. Examples of the former are due to severe photometric changes such as from dark regions

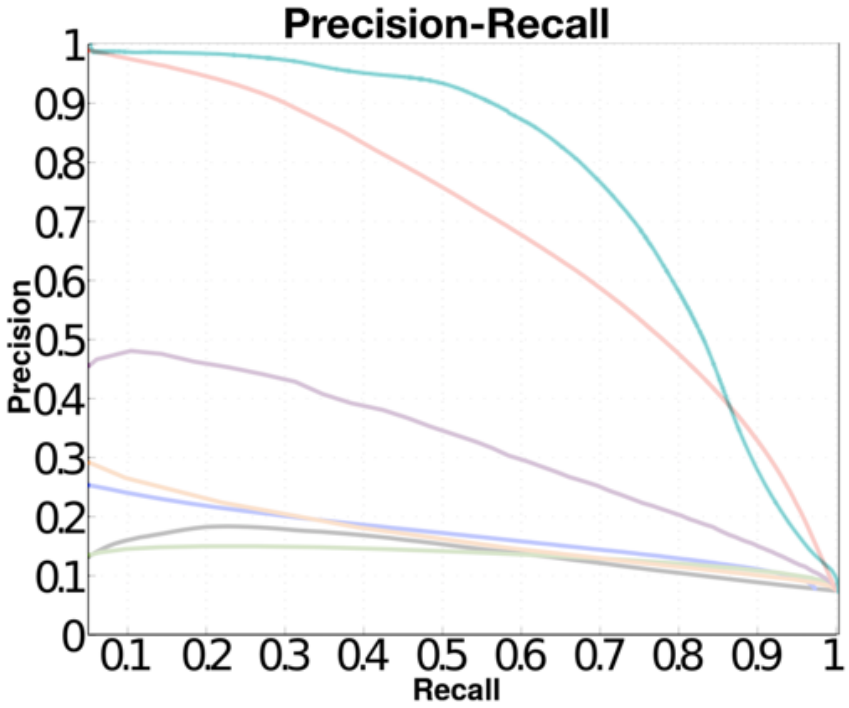


Figure 4.4: Precision-Recall Curve for Binary Change Detection.

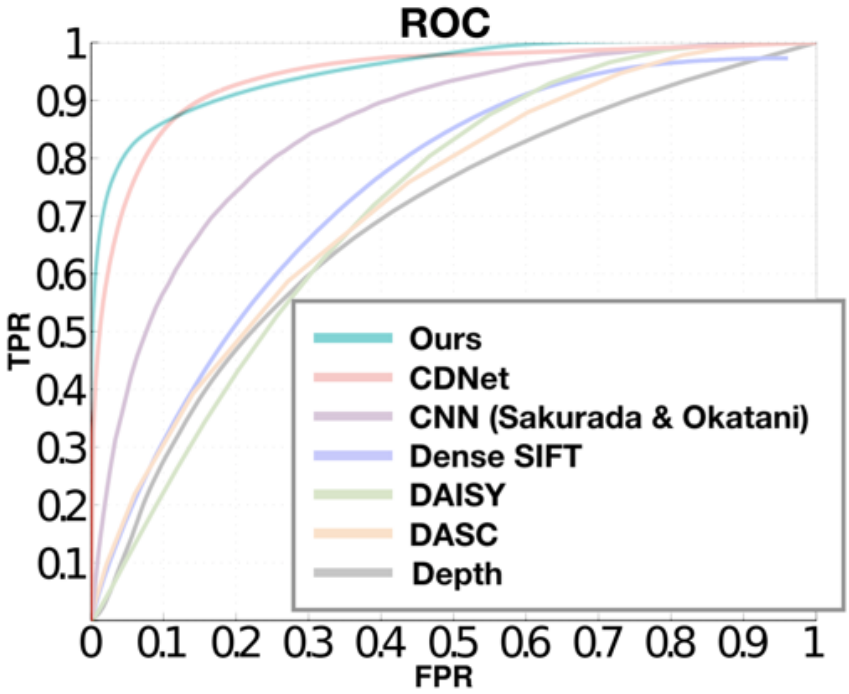


Figure 4.5: ROC Curve for Binary Change Detection.

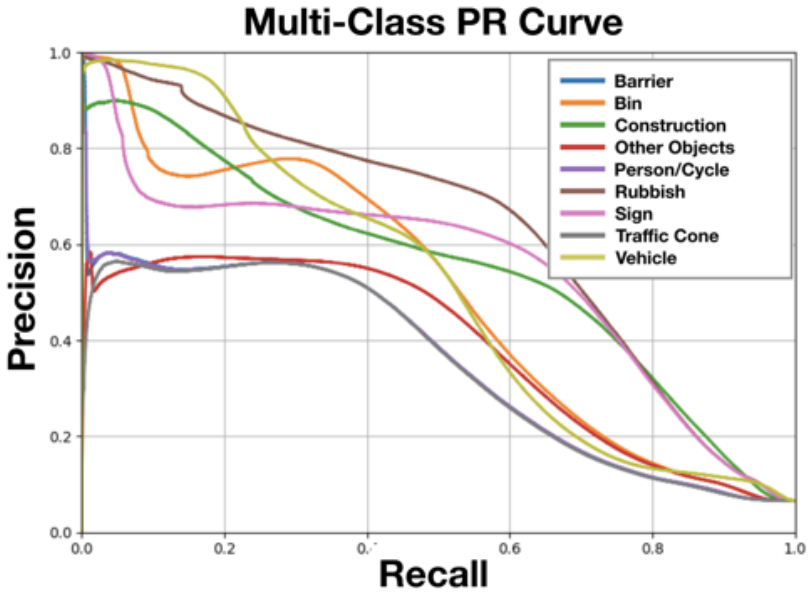


Figure 4.6: Precision-Recall Curve for Semantic Change Detection. Each class curve is generated using a one vs. all approach.

formed by heavy shadows, or from changes with large amounts of transparency with thin features such as see-through fences or bikes. Examples of the latter are due to similar features between like classes, such as rubbish and bins. Several examples are illustrated by Figure 4.9. Upon inspection of several failures of our system, we notice that the VL-CMU-CD dataset has more than a few examples of coarse or incorrect ground truth annotations. Some of the limitations of our network can be attributed to the discrepancies found in the dataset. We highlight some of the issues with the dataset below.

4.6.2 VL-CMU-CD Dataset Limitations

In addition to the change from 11 classes to 10 mentioned in the training section above, we note that while the ground truth images are accurately labeled for binary change, the classes are somewhat coarse for many images, such as those shown in Figure 4.10. There are several examples of both coarsely annotated ground truth and even incorrectly annotated ground truth. In some of these examples, even when the ground truth is incorrectly labeled or coarse, our network is often able to correctly localize and predict the actual change and class respectively. Another issue that arises in this dataset is whether or not the class appeared, or disappeared. In the bottom example of Figure 4.10, we see that *rubbish* is replaced by a *vehicle*. Both are classes that we care about, yet according to the ground truth label, the *rubbish* class is assigned. This is one drawback of the dataset as it currently exists. Note that in the context of LiveMap, if we were to use this method to determine whether or not a hazard is still present, we would not face this issue. The answer would be simple, since we know it existed in the reference image.

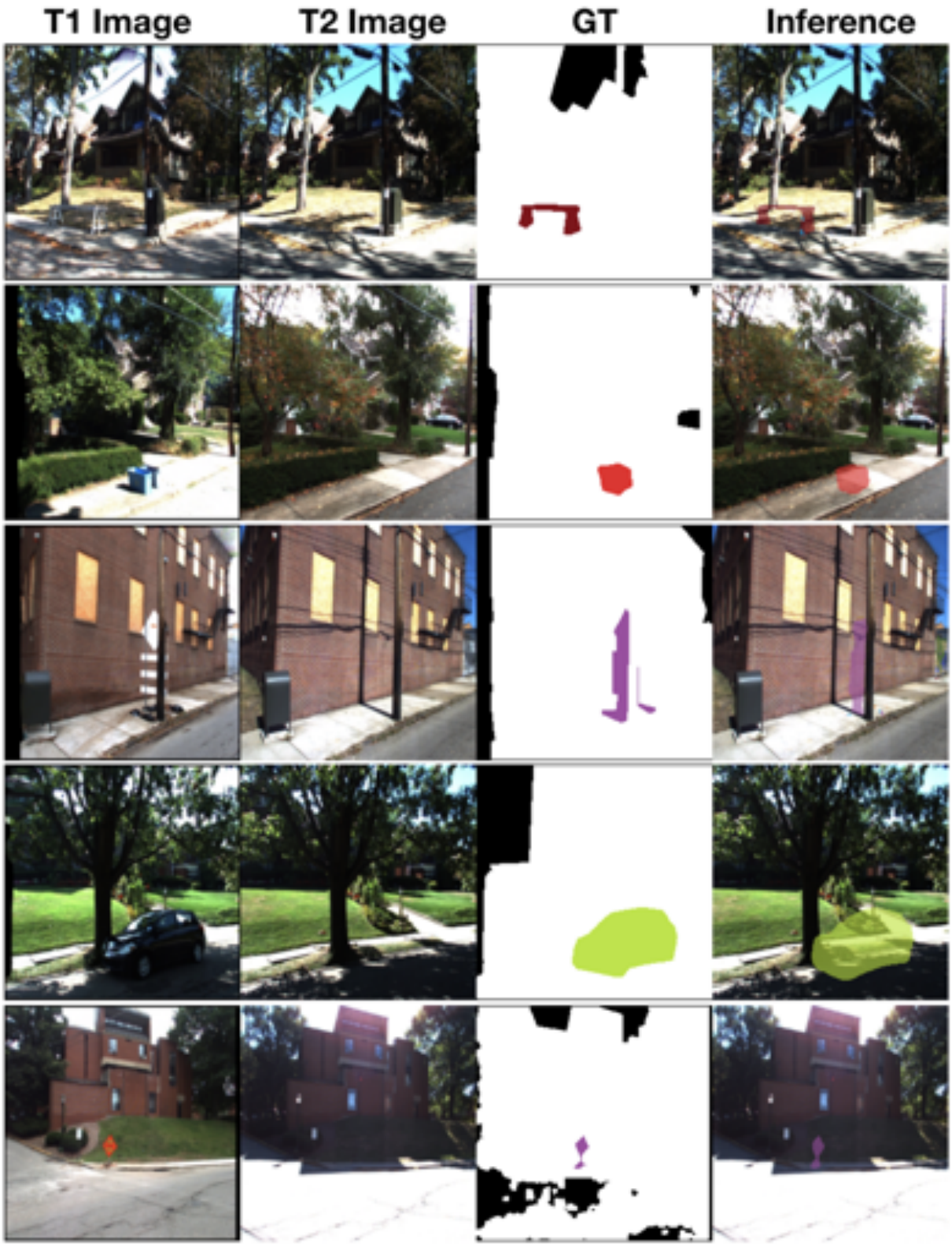


Figure 4.7: Qualitative Performance for Multi-class Change Detection. Images are generated with a False Positive Rate of 0.10. From top to bottom we have: a road barrier (*Barrier*), garbage/recycling *bins*, a traffic *sign*, a *vehicle*, and a road warning *sign*.

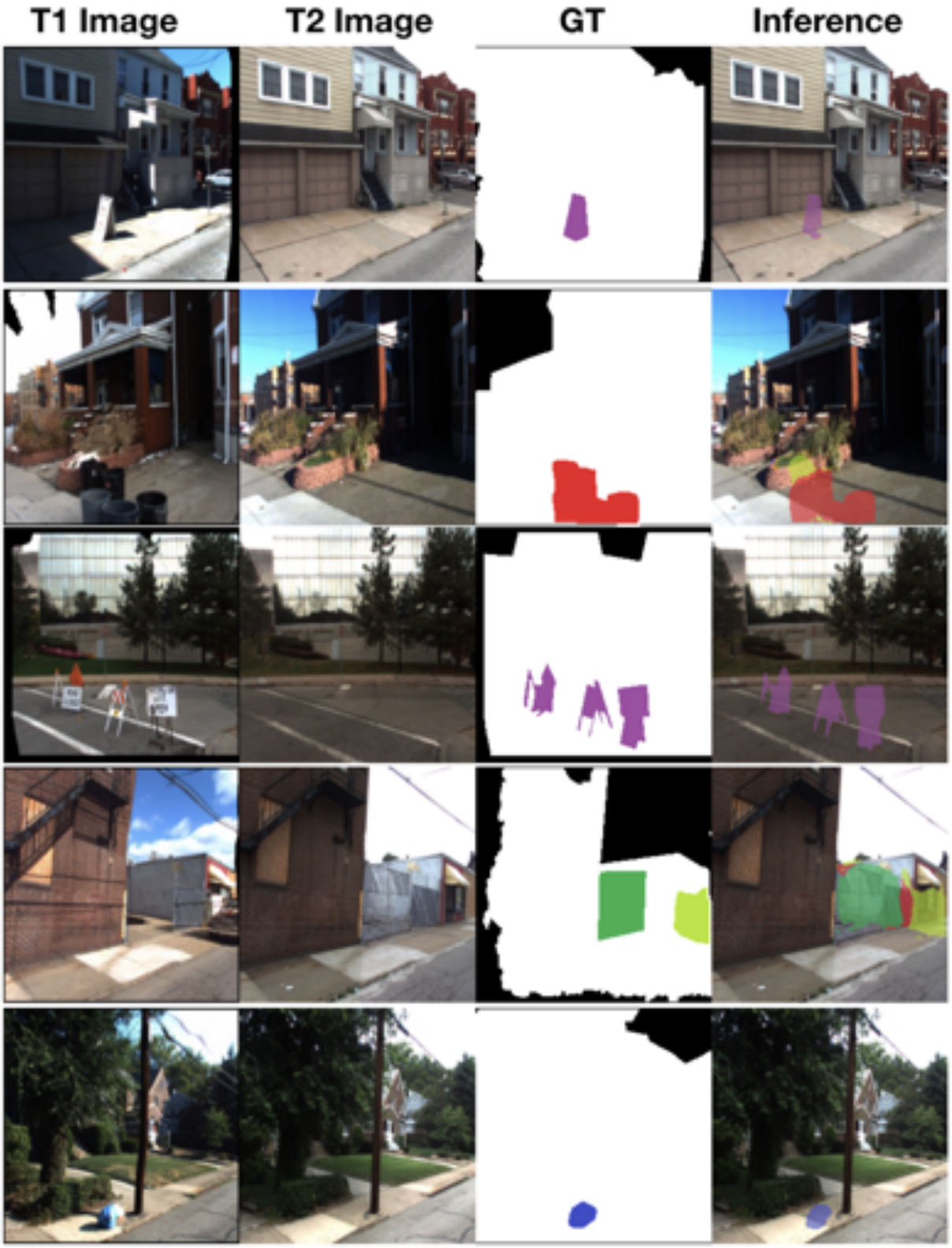


Figure 4.8: Qualitative Performance for Multi-class Change Detection. Images are generated with a False Positive Rate of 0.10. From top to bottom we have: an advertising *sign*, garbage *bins*, road closure *signs*, a fence removed (*other objects*) and a *vehicle*, and *rubbish*.

Figure 4.9: Failure modes for our Semantic Change Detection. Images are generated with a False Positive Rate of 0.10. In the top image, we mis-classify portions of the image. Note the coarse ground truth annotation. In the following example, our network correctly localizes the change, but classifies it as *bins*. However, on closer inspection some of the *rubbish* are indeed *bins*. In the third example, we fail to detect majority of the *bicycle* (there is a very small number of bicycle pixels that are correctly identified but it is difficult to see). Lastly, the *sign* is missed in the last example altogether due to severe photometric differences created by shadows. There is also a false positive change predicted on the left in this example.

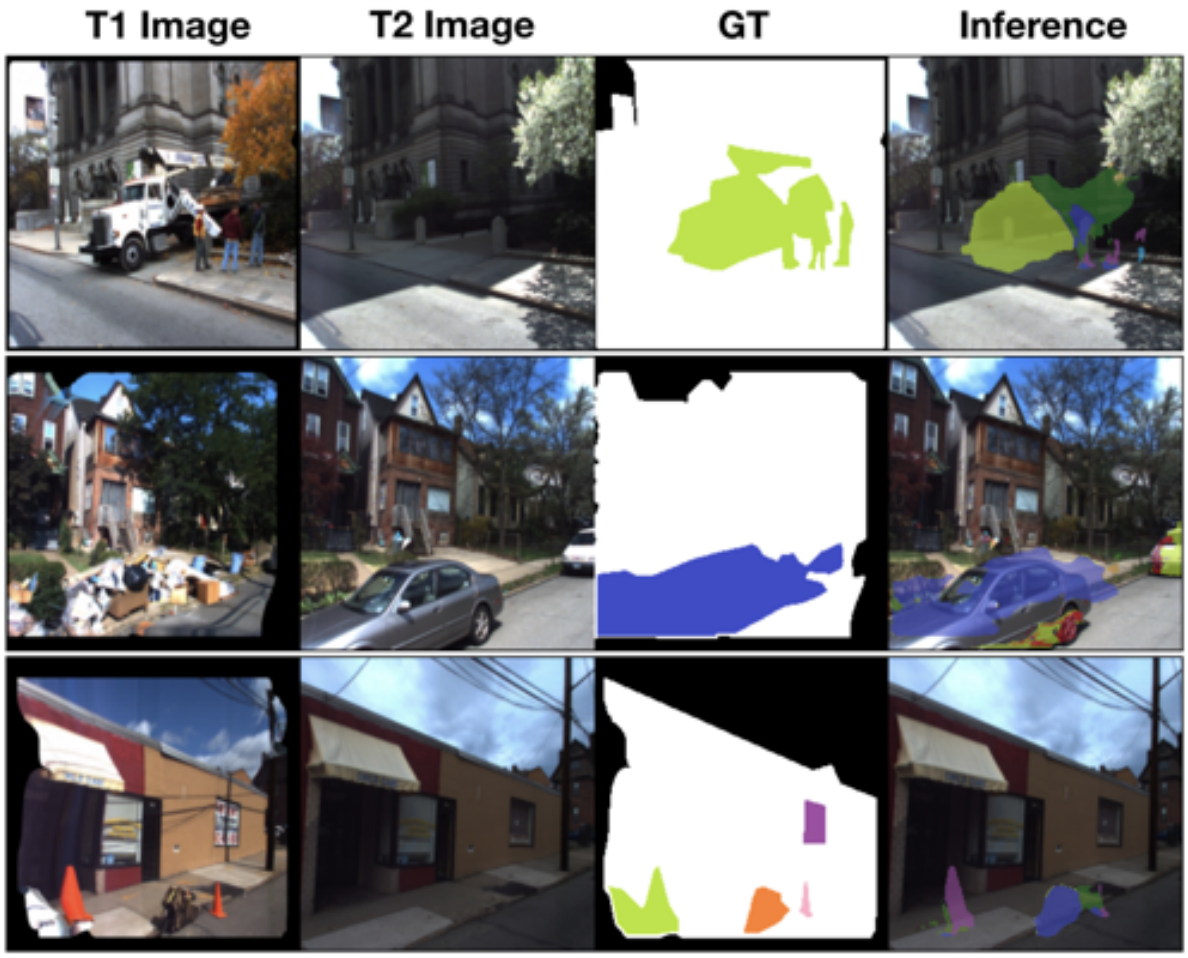


Figure 4.10: Ground truth error examples for VL-CMU-CD dataset. Left two images are the reference and test images. The third image is the change mask, and the last column is our inference at FPR of 0.10. In the top example, the *people* are labeled with the *vehicle* class. In the middle example, the *rubbish* is replaced by a *vehicle*. In this scenario the ground truth is ambiguous. The ground truth could also be labeled with a *vehicle* mask and class. In the bottom example, a *cone* is labeled as a *vehicle*. Note that our system correctly identifies the cone.

Chapter 5

Visual Odometry

In the previous chapters, we focused on deriving semantic meaning from images for live map updates. Another area of improvement to the formulation of LiveMap is to use visual odometry to improve vehicle localization as well as map update localization. In the previous chapters we solely utilized GPS coordinates for the localization of the vehicle. In this chapter, we focus on achieving accurate localization through the development of a novel visual odometry algorithm.

5.1 Motivation

Visual odometry (VO), or the task of tracking camera pose from a stream of images, has received increased attention due to its widespread applications in autonomous vehicles, robotics, and augmented reality. Camera tracking in unknown environments is a very difficult and challenging problem in computer vision. Autonomous vehicles currently rely on very expensive sensors to accurately track position.

While VO has become a more popular area of research, there are still several challenges present. Such challenges are operating in low-texture environments, achieving higher frame rate processing capabilities for increased positional control, and reducing the drift of the trajectory estimate. Any new algorithm must also deal with inherent challenges of tracking camera pose, in particular they must be able to handle the high bandwidth image streams, which requires efficient solutions to extract useful information from such large amounts of data.

5.1.1 Contributions

In this chapter we propose a sparse visual odometry algorithm that efficiently utilizes edges to track the camera motion with state-of-the-art accuracy quantified by low relative pose drift. More formally, we outline our main contributions:

- An edge-direct visual odometry algorithm that outperforms state-of-the-art methods in public datasets.
- We provide experimental evidence that edges are the essential pixels in direct methods through an ablation study.
- We compare our edge method relative to a direct dense method.

Figure 5.1: In contrast to previous direct methods that attempt to minimize the photometric error (bottom left) between reference frame (top left) and input image (top right), we minimize the photometric error of only the edges (bottom right).

- We present key differences on reducing photometric error on edges as opposed to full image intensities.
- We optimize our algorithm with respect to several different types of edges.

5.1.2 Visual Odometry vs. SLAM

Simultaneous localization and mapping (SLAM) algorithms have taken visual odometry algorithms a step further by jointly mapping the environment, and performing optimization over the joint poses and map. Additionally, SLAM algorithms implement loop closure, which enables systems to identify locations which it has visited before and optimize the trajectory by matching feature points against the prior image in memory.

With the success of Bundle Adjustment and loop closure in producing near drift-free results, much of the attention has shifted from the performance of visual odometry algorithms to overall system performance. In reality the two are tightly coupled, and it is very important that visual

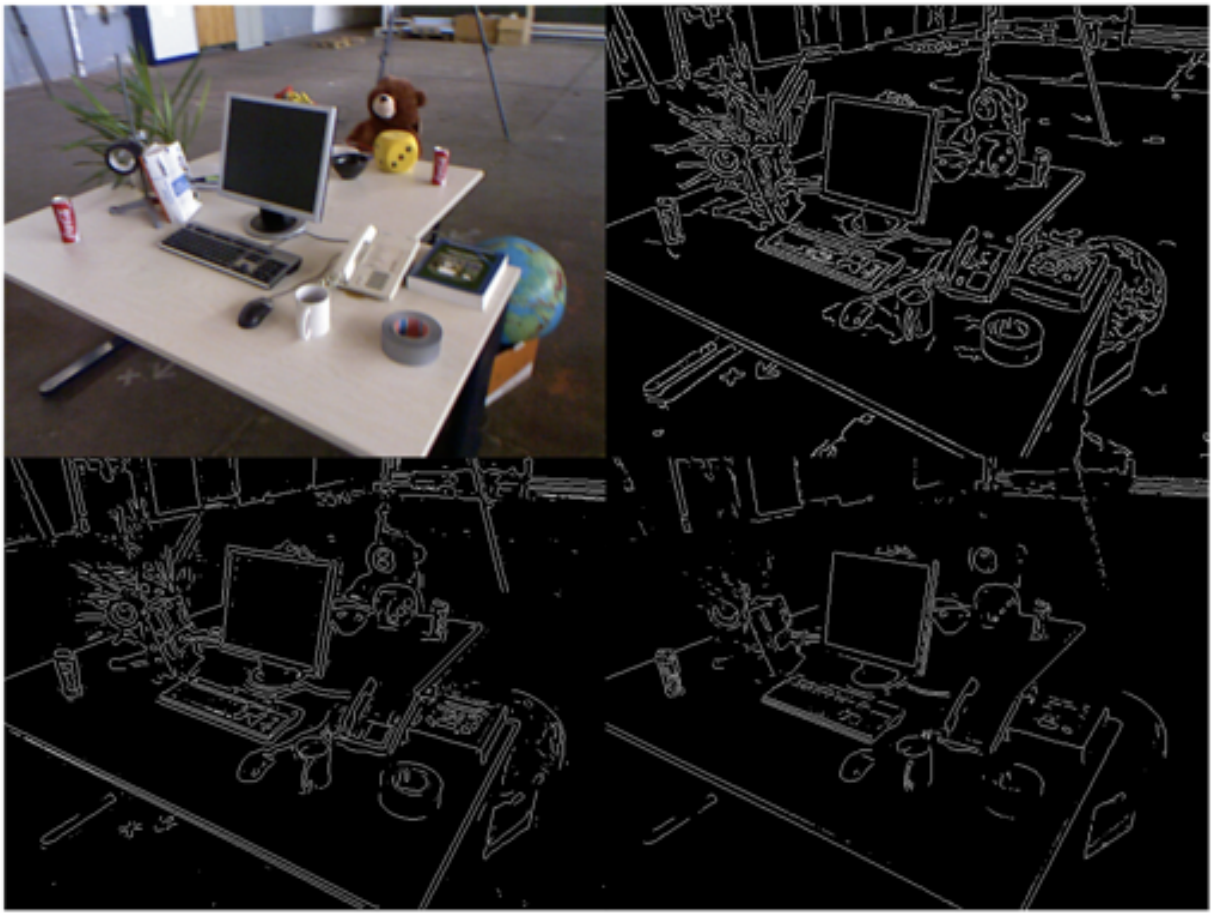


Figure 5.2: Examples of various edge extractions. Top left: Original. Top right: Canny edges. Bottom left: LoG edges. Bottom right: Sobel edges.

odometry provides low-drift pose for two reasons. Firstly, Bundle Adjustment requires a good initialization in order for it to converge to a drift-free solution. Secondly, it is computationally expensive and slow compared to the high frame-rate at which visual odometry operates. For these reasons we focus solely on VO performance in this work, and we show competitive performance even against such SLAM systems.

5.2 Edge-Direct Visual Odometry

5.2.1 Overview

In this section we formulate the edge direct visual odometry algorithm. The key concept behind direct visual odometry is to align images with respect to pose parameters using gradients. This is an extension of the Lucas-Kanade algorithm [6, 69].

At each timestamp we have a reference RGB image and a depth image. When we obtain a new frame, we assume we only receive an RGB image. This enables our method to be extended

Figure 5.3: Edge pyramid for Canny edges. From left to right: Image 1: 640×480, Image 2: 320×240, Image 3: 160×120, Image 4: 80×60. Any pyramid greater than three edge images deep starts to suffer from heavy amounts of aliasing, which led us to cut off our edge pyramid at the third level.

to monocular VO by keeping a depth map and updating at each new time step. Note also that we convert the RGB image into a grayscale image.

The key step of our algorithm is that we then extract edges from the *new* image, and use them as a mask on the *reference* image we are localizing with respect to. We then align the images by iteratively minimizing the photometric error over these edge pixels. The objective is to minimize the nonlinear photometric error

$$r_i(\xi) = \mathcal{I}_2(\tau(x_i, d_i, \xi)) - \mathcal{I}_1(x_i), \tag{5.1}$$

where τ is the warp function that maps image intensities in the second image to image intensities in the first image through a rigid body transform. The warping function $\tau(x_i, d_i, \xi)$ is dependent on the pixel positions x_i, the depth d_i of the corresponding 3D point, and the camera pose ξ. Note that now the pixels we are using are only edge pixels, ie.

$$x_i \in \mathcal{E}(\mathcal{I}_2), \tag{5.2}$$

where $\mathcal{E}(\mathcal{I}_2)$ are the edges of the new image.

5.2.2 Camera Model

In order to minimize the photometric error we need to be able to associate image pixels with 3D points in space. Using the standard pinhole camera model, which maps 3D points to image pixels, we have

$$\pi(P) = \left(\tfrac{f_x X}{Z} + c_x, \quad \tfrac{f_y Y}{Z} + c_y\right)^T, \tag{5.3}$$

where f_x and f_y are the focal lengths and c_x and c_y are the image coordinates of the principal point. If we know the depth then we can find the inverse mapping that takes image coordinates and backprojects them to a 3D point P in homogenous coordinates

$$P = \pi^{-1}(x_i, Z) = \left(\tfrac{x-c_x}{f_x} Z, \quad \tfrac{y-c_y}{f_y} Z, \quad Z, 1\right)^T. \tag{5.4}$$

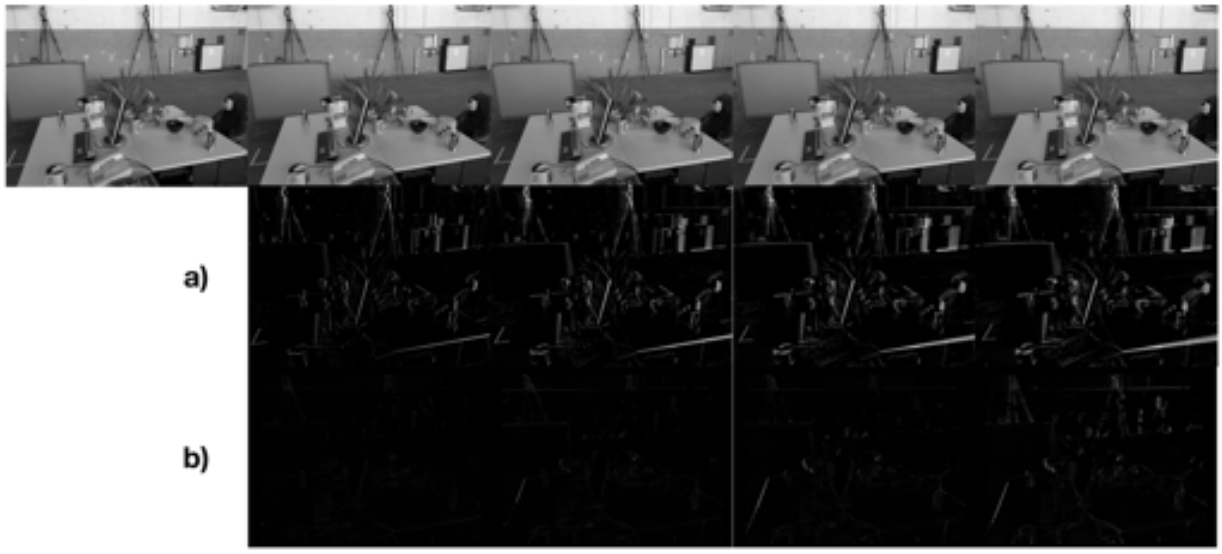


Figure 5.4: In the top row are the original images being localized with respect to the first image. a) All original residuals b) All residuals after only minimizing edge residuals. This shows that minimizing the residuals for just the edge pixels jointly minimizes the residuals for *all* pixels. After 3 images, the minimization starts to become more inaccurate. This is also a function of camera velocity and rotational velocity.

5.2.3 Camera Motion

We are interested in determining the motion of the camera from a sequence of frames, which we model as a rigid body transformation. The camera motion will therefore be in the Special Euclidean Group $SE(3)$. The rigid body transform is given by $T \in SE(3)$

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}, \tag{5.5}$$

where R is a 3×3 rotation matrix and t is a 3×1 translation vector. Since we are performing Gauss-Newton optimization, we need to parameterize camera pose as a 6-vector through the exponential map $T = exp_{\mathfrak{se}(3)}(\xi)$ so that we can optimize over the $SO(3)$ manifold for rotations. At each iteration we can compose the relative pose update $\Delta\xi$ with the previous iteration estimate.

$$\xi^{(n+1)} = \Delta\xi^{(n)} \boxplus \xi^{(n)}, \tag{5.6}$$

where $\Delta\xi \boxplus T = exp_{\mathfrak{se}(3)}(\Delta\xi)T$. We also use a constant motion assumption, where the pose initialization is taken to be the relative pose motion from the previous update, as opposed to initializing with the identity pose. The pose initialization for frame F_i with respect to frame F_k thus can be expressed as

$$\xi_{ki,init} = \xi_{k,i-1} \boxplus \xi_{i-2,i-1}. \tag{5.7}$$

. Experimentally we have found that this greatly improves performance by providing the system with an accurate initialization such that it can converge to a low-error solution.

5.2.4 Robust Gauss-Newton on Edge Maps

Similar to other direct methods, we employ a coarse-to-fine approach to Gauss-Newton minimization to avoid false convergence. The selection of the image pyramid scheme has a large effect on the system performance, and must be chosen carefully. Some systems such as [26] report using up to six levels, while [50] report using four levels. Simply extending these large pyramid sizes to edge maps causes the system to fail to converge to the correct solution. This is due to the effects of aliasing. A much smaller pyramid size is required.

We found that three levels worked well for the original 640×480 resolution. Using additional levels caused the system to fail due to edge aliasing effects which is illustrated in Figure 5.3, which shows the same edge image at different levels of the pyramid. After level three, it becomes unrecognizable. For this reason, we recommend using images no smaller than 160×120 in resolution.

A common approach in direct methods is to incorporate a weighting function that increases robustness to outliers when solving the error function. We use an iteratively re-weighted residual error function that we minimize with Gauss-Newton. We found that iteratively re-weighting using Huber weights worked quite well for our application, following the work of [26]. The Huber weights are defined as

$$w_i(r_i) = \begin{cases} 1, & r_i \leq k \\ \frac{k}{|r_i|}, & r_i > k \end{cases}. \tag{5.8}$$

The error function now becomes

$$E(\xi) = \sum_i w_i(\xi) r_i^2(\xi). \tag{5.9}$$

Our goal is to find the relative camera pose that minimizes this function

$$\arg\min_{\xi} E(\xi) = \arg\min_{\xi} \sum_i w_i(\xi) r_i^2(\xi). \tag{5.10}$$

In order to minimize this nonlinear error function with Gauss-Newton, we must linearize the equation. We can then solve this as a first-order approximation by iteratively solving the equation

$$\Delta\xi^{(n)} = -(J^T W J)^{-1} J^T W r(\xi^{(n)}), \tag{5.11}$$

where W is a diagonal matrix with the weights, and the Jacobian J is defined as

$$J = \nabla\mathcal{I}_2 \frac{\partial \pi}{\partial P} \frac{\partial P}{\partial T} \frac{\partial T}{\partial \xi}, \tag{5.12}$$

and $\nabla\mathcal{I}_2$ is the image gradient of the new image. We can then iteratively update the relative pose with Equation 5.6.

Note that we use the inverse-composition [6] formulation such that we do not have to recompute the Jacobian matrix every iteration. This is what makes this algorithm extremely efficient, as shown in [6].

5.2.5 Optimizing over Edge Points

We present the theory of selecting and incorporating edge points in the formulation, and provide some insight on why it is so effective in implementation. For edge selection process, note that we have two images, a reference and a new image, and therefore two sets of edges. We wish to avoid the problems that arise from using both sets, namely there will be a different number of edge pixels, and dealing with this through matching algorithms is inefficient and error-prone. We use a more elegant solution, which is to use the edges of the new image as a mask on the first image.

This initialization causes the mask to select pixels in the reference image that are slightly off from the reference image edges, assuming the camera has moved. At each iteration, we follow a gradient from this position towards a point that reduces photometric error. By definition, edges are regions of large photometric variation on either side. Intuitively we argue that the optimization should therefore converge and settle at the correct edge. To summarize, we initialize the edge mask to an offset position from the reference image's edges, and iteratively force these edge pixels to overlap with the reference edges. In doing this we achieve a highly accurate relative pose.

5.2.6 Keyframe Selection

Another implementation detail has to do with keyframes. Frame-to-frame alignment is inherently noisy and prone to accumulate drift. To mitigate this, VO algorithms often select a key-frame which is used as the reference image for multiple new frames. The error accumulation is decreased by comparing against fewer reference frames, which directly results in a smaller error stackup.

There have been several strategies for selecting keyframes. The selection of keyframes is dependent on the type of VO algorithm being used. Feature-based methods such as [73] usually impose the restriction that a significant number of frames to pass, on the order of tens of frames.

In [51] the authors summarize several common approaches that direct methods use for creating a new keyframe: every n frames, after a certain relative pose threshold has been met, the variance of the error function exceeds a threshold, or the differential entropy of the covariance matrix reaches a threshold. However, each metric is not without its problems.

Furthermore, the performance of the tracking degrades the further apart the baselines. Figure 5.4 demonstrates this phenomena, in which the residuals from five consecutive frames with respect to the first frame are shown. We observe that in general after 4 frames, the residuals become harder to minimize for most sequences. Note that this is a function of camera motion. We make the assumption that this camera tracking will be used for moderate motion and select an every n frames approach.

5.3 Experiments

We evaluate our system using the TUM RGB-D benchmark [104] , which is provided by the Technical University of Munich. The benchmark has been widely used by various SLAM and

Relative Pose Error (RPE) [m/s]											
Seq.	ours Canny$_{KF}$	ours Canny$_{FF}$	ours LoG$_{FF}$	ours Sobel$_{FF}$	ours SE$_{FF}$	REVO[93] SE$_{FF}$	REVO[93] SE$_{KF}$	DSLAM[51] ICP+Gray	ORB2[72] Feat.	SLAM[25] Feat.	D-EA[59] Canny
fr1/xyz	0.02228	0.02768	0.02821	0.02712	0.03289	0.03202	0.01957	0.02661	**0.01470**	0.04193	0.04942
fr1/rpy	-	0.03126	**0.02714**	0.03694	0.03041	0.03553	0.04037	0.04865	0.03221	0.07028	0.16150
fr1/desk	**0.02664**	0.03022	0.03022	0.03598	0.03056	0.07800	0.22196	0.04429	0.06178	0.05346	0.10654
fr1/desk2	-	**0.04387**	0.04953	0.05566	0.04490	0.07056	0.06703	0.05722	0.06535	0.06955	0.20117
fr1/room	-	0.04830	0.06239	0.05240	0.05006	0.04816	**0.04272**	0.06427	0.07081	0.06666	0.21649
fr1/plant	-	**0.02736**	0.02752	0.04171	0.05006	0.03063	0.02381	0.04362	0.04218	0.03789	0.34099
fr2/desk	**0.01237**	0.01375	0.01375	0.01800	0.03021	0.01426	0.02453	0.03248	0.03067	0.01400	0.09968
Absolute Trajectory Error (ATE) [m]											
fr1/xyz	0.04567	0.04478	0.04461	0.05260	0.04115	0.09011	0.05375	0.05760	**0.00882**	0.01347	0.13006
fr1/rpy	-	0.05561	0.03982	0.04795	0.04983	0.08933	0.07684	0.16341	0.08090	**0.02874**	0.14822
fr1/desk	0.03133	0.05387	0.05358	0.05977	0.04802	0.18648	0.54789	0.18251	0.09091	**0.02583**	0.16376
fr1/desk2	-	0.06798	0.08384	0.08189	0.06271	0.16866	0.18163	0.18861	0.10090	**0.04256**	0.44886
fr1/room	-	0.27382	0.34505	0.31586	0.34167	0.30594	0.28897	0.21559	0.20282	**0.10117**	0.60361
fr1/plant	-	0.07559	0.06708	0.09975	0.06560	0.07300	**0.05623**	0.12216	0.07234	0.06388	0.56927
fr2/desk	0.12540	0.16664	0.18830	0.19074	0.27464	0.32902	0.09590	0.46796	0.38657	**0.09505**	0.94546

Table 5.1: Comparison of the performance of our system using three different types of edges. Blue denotes best performing frame-to-frame VO, excluding SLAM or keyframe systems. **Bold** denotes best performing system overall. A dashed line indicates that using keyframes did not improve performance.

VO algorithms to benchmark their accuracy and performance over various test sequences. Each sequence contains RGB images, depth images, accelerometer data, as well as groundtruth. The camera intrinsics are also provided. Groundtruth was obtained by an external motion capture system through triangulation, and the data was synchronized.

There are several challenging datasets within this benchmark. Each sequence ranges in duration, trajectory, and translational and rotational velocities. We follow the work of [93] which uses seven sequences to benchmark their system performance so to achieve a direct comparison with other methods.

5.3.1 Evaluation Metrics

We use the Relative Pose Error (RPE) and Absolute Trajectory Error (ATE) to evaluate our system. The Relative Pose Error is proposed for evaluation of drift for VO algorithms in [104]. It measures the accuracy of the camera pose over a fixed time interval Δt

$$RPE_t = (Q_t^{-1} Q_{t+\Delta t})(P_t^{-1} P_{t+\Delta t}), \tag{5.13}$$

where $Q_1 \dots Q_n \in SE(3)$ are the camera poses associated with the groundtruth trajectory and $P_1 \dots P_n \in SE(3)$ are the camera poses associated with the estimated camera trajectory. Similarly the Absolute Trjectory Error is defined as

$$ATE_t = Q_i^{-1} S P_i, \tag{5.14}$$

where poses Q and P are aligned by the rigid body transformation S obtained through a least-squares solution.

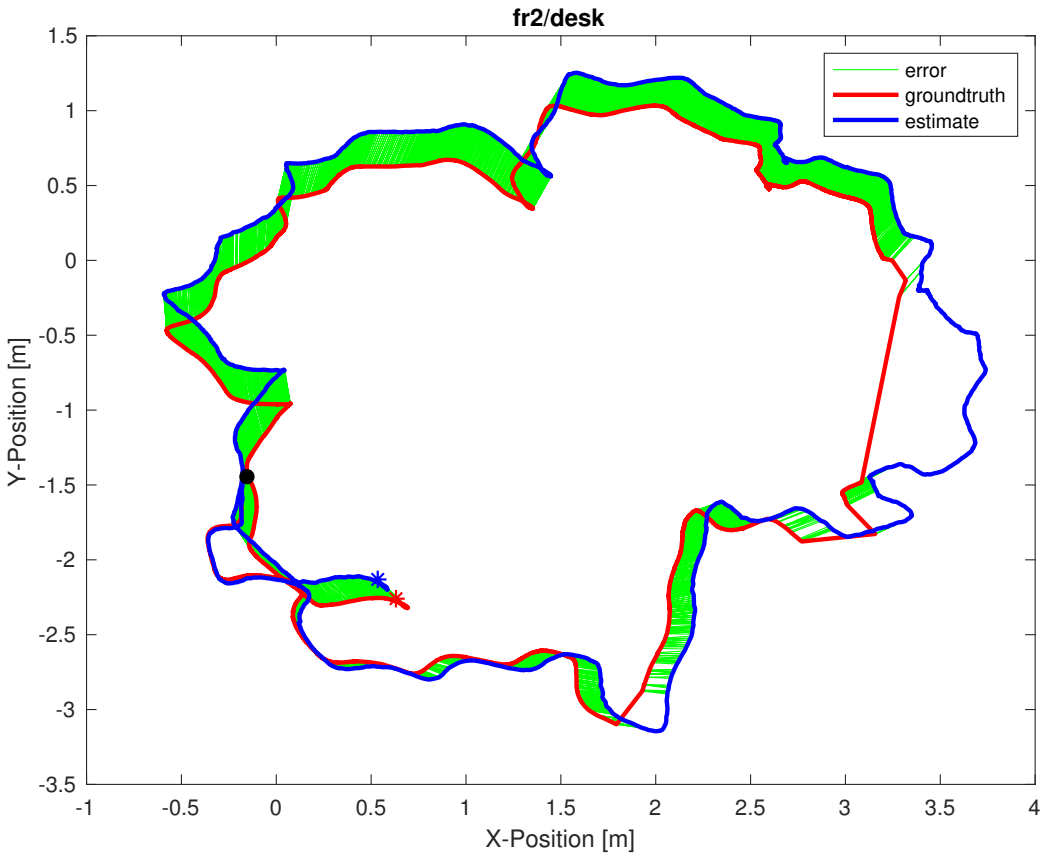


Figure 5.5: XY cross-section of our estimated trajectory compared with ground truth. The error is shown in green. The start position is shown as a black dot, while the final positions are shown as colored dots corresponding to the trajectory. Areas without green indicate missing groundtruth data from sequence.

A common practice has been to use the RMSE value of both the RPE and ATE, as RMSE values are a more robust metric that gives more weight to outliers as compared with the mean or median values. Thus the RMSE is a much more stringent performance metric to benchmark system drift.

Following the example set by [28, 50, 107], we provide the RMSE camera pose drift over several sequences of the dataset. As first pointed out in [50], choosing too small of a Δt creates erroneous error estimates as the ground truth motion capture system has finite error as well. Too large of a value leads to penalizing rotations more so at the beginning than rotations towards the end [104]. Therefore, a reasonably sized Δt needs to be chosen. We use a Δt of 1s to achieve direct comparison with other methods.

5.3.2 Results on the TUM RGB-D Benchmark

We compare the performance of our algorithm using four different edge extraction algorithms, namely Canny, LoG, Sobel, and Structured Edges. We compare to other methods using frame-to-frame tracking for all variants. We selected Canny to perform keyframe tracking due to its consistent accuracy. Although all of the edge types performed well on the sequences, Canny edges performed the best overall on average. Note that we used automatic thresholding as opposed to REVO [93] which used fixed threshold values, which introduces a dependency on photometric consistency. Since we utilize automatic thresholding, our system is more robust to photometric variations across frames. See Figure 5.2 for examples of edge extractions.

From our experiments we observed that edge-direct VO is highly accurate in frame-to-frame tracking, despite the inherent accumulation of drift in such a scheme that does not utilize keyframes. In terms of RPE, our frame-to-frame variants perform better than or in worst case as well as REVO, an edge-based method which uses the distance transform on edges. Our method also outperforms ORB-SLAM2 run in VO mode for all sequences, except on $fr1/xyz$. This is a result of ORB-SLAM2 keeping a local map, and in this particular sequence the camera keeps the majority of the initial scene in view at all times. We confirmed this hypothesis by turning off the local mapping, at which case we outperform it on this sequence as well. Our results are shown in Table 5.1. In terms of ATE, we again perform well across all non-SLAM algorithms. Even though we do not use any Bundle Adjustment or global optimization as employed by RGBD-SLAM [51], we perform competitively over all sequences with such systems.

We provide plots of the edge-direct estimated trajectories over time compared to groundtruth in Figure 5.7. Our estimated trajectory closely follows that of the groundtruth. In Figure 5.5 we show the edge-direct estimated trajectory along the XY plane, along with the error between our estimate and groundtruth.

5.3.3 Ablation Study

In order to experimentally demonstrate the effect of using edge pixels we perform an ablation study. This two-fold ablation study demonstrates the relative efficacy between optimizing over edge pixels compared with optimizing over the same number of randomly chosen pixels, and additionally demonstrates the stability of using edge pixels. We randomly select a fraction of the edge pixels to use, and compare it to our system randomly selecting the same number of pixels from the entire image. We average over 5 runs to account for variability. All parameters are identical for both methods. Additionally, for these tests we utilize keyframes as well as dropping the constant motion assumption. This forces the system to rely on the optimization more heavily, and provides a better measurement of the quality of convergence. We additionally record the latency of our system per frame. Operating on edge pixels is more accurate, while additionally enabling $\sim$50 fps on average on an Intel i7 CPU. Note that at our optimization settings, a dense method is far from real-time.

Since we use the Lucas-Kanade Inverse Compositional formulation we expected our algorithm to be linear time complexity with the number of pixels used. We confirm this experimentally as well. Refer to Figure 5.6 for both the ablation study and timing measurements. We save approximately 90% computation on average by using edge pixels compared to using all pixels.

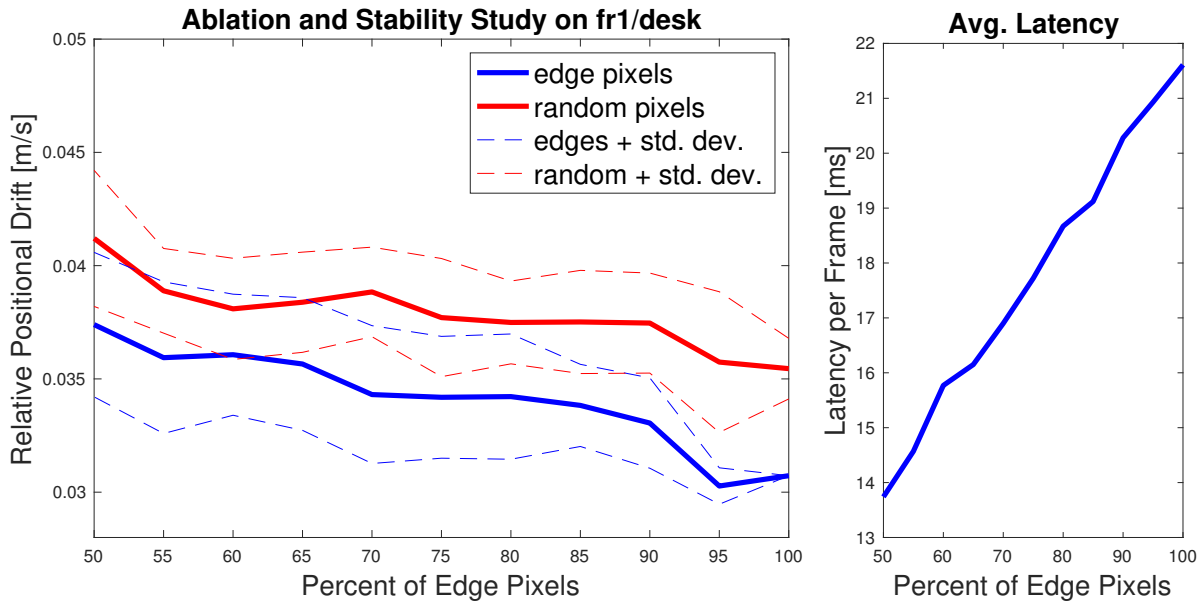


Figure 5.6: Left: Ablation study. Right: Frame-to-frame latency using edges.

Note that for stability of edge pixels, the Kinect sensor used in the sequences filters out unstable points in its depth map, and from qualitative inspection still leaves a large number of reliable edge pixels. This is confirmed via the relative stability of selected edge pixels compared to all pixels as well. This ablation study further supports our claim that edge pixels are essential for robust and accurate camera tracking.

5.4 Discussion

Our edge-direct VO algorithm performs well across all sequences compared to other state-of-the-art methods. The trajectory in Figure 5.5 shows accurate camera tracking in a sequence that is 99 seconds long, and travels over 18 m without the use of Bundle Adjustment or loop closure. Note that our algorithm would perform even better if coupled with such global optimization methods, as our VO would initialize the algorithms closer to the correct solution compared with other algorithms. Such an increase in accuracy can enable SLAM systems to rely less heavily on computationally expensive global optimizations, and perhaps run these threads less frequently. Note that in this figure, the regions that are missing green regions are due to missing groundtruth data in the sequence. The estimated trajectory over time in Figure 5.7 shows remarkably accurate results as well.

It is important to note that even though we explicitly only minimize the photometric error for edge pixels, Figure 5.4 shows that we simultaneously minimize the residuals for all pixels. This is an important observation, as it supports the claim that minimizing the residuals of edge pixels is the minimally sufficient objective. Moreover, the ablation study supports the claim that minimizing the photometric residuals for just the edge pixels provides less pixels to iterate over while enabling accurate tracking.

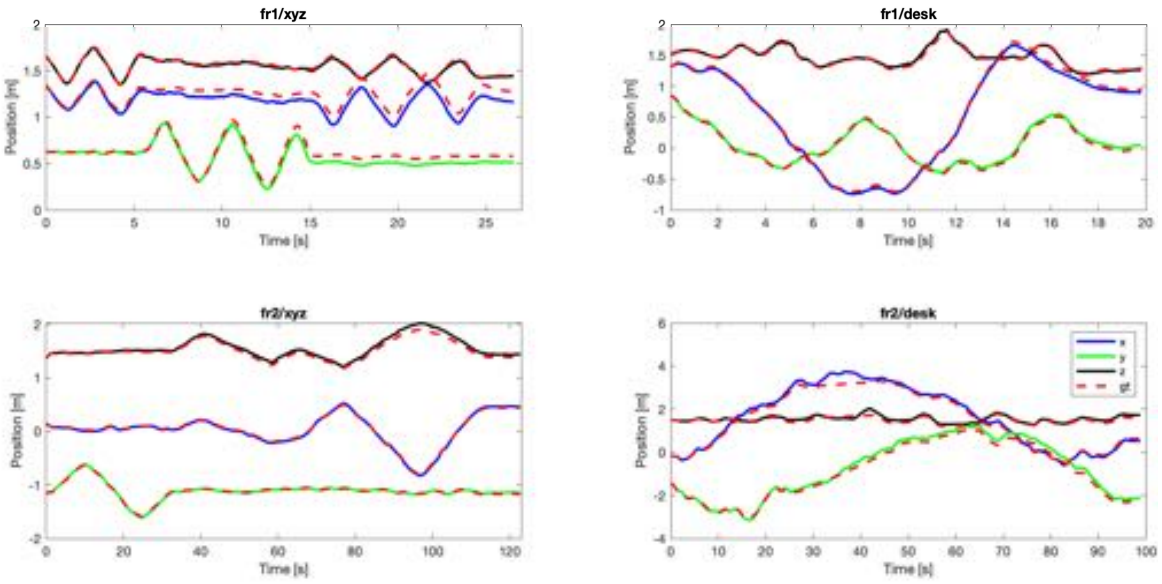


Figure 5.7: Shown is our estimated trajectory for four sequences. Each sequence plots the trajectory in solid colors corresponding to the axis. Groundtruth is shown as a red dotted line for all axes. As can be seen our estimates closely match that of the ground truth. Note that for the sequence fr2/desk, there is no ground truth during the interval at approximately 31-43 seconds, which is why there appears to be a straight line in groundtruth trajectory.

It is interesting to note that utilizing keyframes did not help the system improve on many of the sequences once we added the constant motion assumption. Prior to adding this camera motion model, utilizing keyframes helped significantly.

Chapter 6

Conclusion

We have proposed a system architecture, LiveMap, that automates the detection and reporting of road hazard information utilizing in-vehicle compute and recent advances in computer vision. We have built and demonstrated a prototype system using both powerful and modest in-vehicle computers coupled with edge computing services. Both variants are able to detect and report potholes with no human involvement. Furthermore, we reduce the bandwidth consumed by such a system by over twenty-fold compared to video streaming to the cloud for processing.

To help improve our system, we developed a change detection network that is able to accurately determine semantic pixel level change between two images. We have tested our change detection system extensively on the VL-CMU-CD dataset. We benchmark and compare its performance to several other systems. We not only surpass prior systems, but also extend them by enabling multi-class change detection.

In the last chapter we presented an edge-direct visual odometry algorithm that can enable more accurate localization of the vehicle as well as map updates. Our algorithm determines an accurate relative pose between two frames by minimizing the photometric error exclusively of edge pixels. We demonstrate experimentally that minimizing the edge residuals jointly minimizes the residuals over the entire image. This minimalist representation reduces computation required by operating on all pixels, and also results in more accurate tracking. We benchmark its performance on the TUM RGB-D dataset where it achieves state-of-the-art performance as quantified by low relative pose drift and low absolute trajectory error.

6.1 Future Work

Several potential areas of future research and continuation exist for LiveMap. While we developed semantic change detection and visual odometry algorithms to improve the overall system, we leave the integration of such features for future work. An additional functionality improvement for LiveMap includes expanding the number of vehicles, such that additional features can be explored and deployed on multiple live vehicles. Another avenue is to expand the classes of hazards that LiveMap can detect. Data is often a limiting factor for the performance of vision algorithms, and this is true with LiveMap. The hazard detection accuracy would see a performance increase with increased training set sizes. Moreover, instead of using Faster R-CNN, future ver-

sions of LiveMap may move towards instance segmentation, such as with Mask R-CNN. Another functionality addition could be for emergency situations, such as finding stolen cars or even a lost child. To perform such a task, vehicles could analyze both current and past data within a region of interest when prompted by the Zone Cloudlet.

In regards to the change detection pipeline, the system could improve by moving towards *Panoptic Change Detection*, which would be analogous to the newly created panoptic segmentation task. In this new task (possibly first termed in this thesis), change would be determined at instance level, and background would be on the semantic level. This would be useful in cases where we care about the distinction between the boundaries of multiple overlapping objects, which is common with vehicles, signs, etc.

One requirement for the visual odometry algorithm as outlined in Chapter 5 is that it requires an accurate and dense depth map. There are such systems, however they are either achieved in real-time via non-optimal local solutions or through computationally expensive global solutions, which have time complexity $O(n^2)$ where n is the number of pixels. For large n, such algorithms are prohibitively expensive. One research area would be to explore the possibility of a more optimal solution.

Bibliography

[1] Pablo F Alcantarilla, Simon Stent, German Ros, Roberto Arroyo, and Riccardo Gherardi. Street-view change detection with deconvolutional networks. *Autonomous Robots*, 42(7): 1301–1322, 2018. 2.2.5, 2.6.2, 4.1, 4.3, 4.1, 4.6

[2] Microsoft Azure. URL https://azure.microsoft.com. 2.4

[3] Hernán Badino, D Huber, and Takeo Kanade. Visual topometric localization. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 794–799. IEEE, 2011. 2.6.2

[4] Hernán Badino, Daniel Huber, and Takeo Kanade. Real-time topometric localization. In *2012 IEEE International Conference on Robotics and Automation*, pages 1635–1642. IEEE, 2012. 2.6.2

[5] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017. 2.2, 2.2.1, 4.4

[6] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *Int. J. Comput. Vision*, 56(3):221–255, February 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000011205.11775.fd. URL https://doi.org/10.1023/B:VISI.0000011205.11775.fd. 5.2.1, 5.2.4

[7] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010. 2.1

[8] Neal E. Boudette. Biggest spike in traffic deaths in 50 years? blame apps. URL https://www.nytimes.com/2016/11/16/business/tech-distractions-blamed-for-rise-in-traffic-fatalities.html. 1

[9] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. Coco-stuff: Thing and stuff classes in context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1209–1218, 2018. 2.2.4

[10] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov 1986. ISSN 0162-8828. doi: 10.1109/TPAMI.1986.4767851. 2.6.3

[11] D. B. A. Chacra and J. S. Zelek. Fully automated road defect detection using street view images. In *2017 14th Conference on Computer and Robot Vision (CRV)*, pages 353–360, May 2017. doi: 10.1109/CRV.2017.50. 2.6.1

[12] Zhuo Chen, Wenlu Hu, Junjue Wang, Siyan Zhao, Brandon Amos, Guanhang Wu, Kiryong Ha, Khalid Elgazzar, Padmanabhan Pillai, Roberta Klatzky, et al. An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 14. ACM, 2017. 2.4

[13] Kevin Christensen, Christoph Mertz, Padmanabhan Pillai, Martial Hebert, and Mahadev Satyanarayanan. Towards a distraction-free waze. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 15–20. ACM, 2019. 1.2, 3.1

[14] Google Cloud. URL https://cloud.google.com/. 2.4

[15] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016. 2.2.5

[16] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995. 2.2.2

[17] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3150–3158, 2016. 2.2.3

[18] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016. 2.2.2

[19] Jifeng Dai, Haozhi Qi, and Yi Li. Translation-aware fully convolutional instance segmentation. 2016. 2.2.3

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 2.1, 2.2.1, 2.2.5

[21] Piotr Dollár and C. Lawrence Zitnick. Structured forests for fast edge detection. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ICCV '13, pages 1841–1848, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-1-4799-2840-8. doi: 10.1109/ICCV.2013.231. URL http://dx.doi.org/10.1109/ICCV.2013.231. 2.6.3

[22] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul): 2121–2159, 2011. 2.1

[23] Ethan Eade and Tom Drummond. Edge landmarks in monocular slam. *Image Vision Comput.*, 27(5):588–596, April 2009. ISSN 0262-8856. doi: 10.1016/j.imavis.2008.04.012. URL http://dx.doi.org/10.1016/j.imavis.2008.04.012. 2.6.3

[24] Eclipse Paho. URL https://www.eclipse.org/paho/. 3.2.1

[25] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3-d mapping with an rgb-d

camera. *IEEE Transactions on Robotics*, 30(1):177–187, Feb 2014. ISSN 1552-3098. doi: 10.1109/TRO.2013.2279412. 5.2.6

[26] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*, September 2014. 2.6.3, 5.2.4

[27] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, March 2018. 2.6.3

[28] Jakob Engel, Jürgen Sturm, and Daniel Cremers. Semi-dense visual odometry for a monocular camera. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ICCV '13, pages 1449–1456, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-1-4799-2840-8. doi: 10.1109/ICCV.2013.183. URL http://dx.doi.org/10.1109/ICCV.2013.183. 5.3.1

[29] Envrmnt. URL https://www.envrmnt.com/. 2.4

[30] Jakob Eriksson, Lewis Girod, Bret Hull, Ryan Newton, Samuel Madden, and Hari Balakrishnan. The pothole patrol: Using a mobile sensor network for road surface monitoring. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, MobiSys '08, pages 29–39, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-139-2. doi: 10.1145/1378600.1378605. URL http://doi.acm.org/10.1145/1378600.1378605. 2.5, 2.6.1

[31] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. 2.2.2, 2.2.5

[32] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2012. 2.2.1

[33] Leifer Law Firm. Is the waze driving app too distracting? URL https://www.leiferlaw.com/waze-driving-app-distracting/. 1

[34] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012. 2.2.5

[35] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 2.2.2

[36] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. 2.2.2

[37] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*, 2010. 2.1

[38] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. 2009. 2.2.5

[39] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, Dec 2015. doi: 10.1109/ICCV.2015.123. 2.1

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015. 2.2.2

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2.2.1, 2.6.2

[42] 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. 2.2.3, 2.4

[43] Seunghoon Hong, Hyeonwoo Noh, and Bohyung Han. Decoupled deep neural network for semi-supervised semantic segmentation. In *Advances in neural information processing systems*, pages 1495–1503, 2015. 2.2.1

[44] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861. 3.2.4

[45] Wenlu Hu, Ziqiang Feng, Zhuo Chen, Jan Harkes, Padmanabhan Pillai, and Mahadev Satyanarayanan. Live synthesis of vehicle-sourced data over 4G LTE. In *Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '17, pages 161–170, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5162-1. doi: 10.1145/3127540.3127543. URL http://doi.acm.org/10.1145/3127540.3127543. 2.6.1, 3.1, 3.3.1

[46] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017. 2.2.2

[47] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen K. Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. CarTel: A Distributed Mobile Sensor Computing System. In *4th ACM SenSys*, Boulder, CO, November 2006. 2.6.1

[48] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017. 2.2.1

[49] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0716-1. doi: 10.1145/2047196.2047270. URL

http://doi.acm.org/10.1145/2047196.2047270. 2.6.3

[50] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for rgb-d cameras. In *2013 IEEE International Conference on Robotics and Automation*, pages 3748–3754, May 2013. doi: 10.1109/ICRA.2013.6631104. 5.2.4, 5.3.1

[51] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgb-d cameras. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2100–2106, Nov 2013. doi: 10.1109/IROS.2013.6696650. 2.6.3, 5.2.6, 5.3.2

[52] Seungryong Kim, Dongbo Min, Bumsub Ham, Seungchul Ryu, Minh N Do, and Kwanghoon Sohn. Dasc: Dense adaptive self-correlation descriptor for multi-modal and multi-spectral correspondence. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2103–2112, 2015. 4.1, 4.6

[53] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015. 2.1, 4.4.1

[54] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic segmentation. *arXiv preprint arXiv:1801.00868*, 2018. 2.1, 2.2.4

[55] G. Klein and D. W. Murray. Improving the agility of keyframe-based SLAM. In *Proc 10th European Conf on Computer Vision, Marseille, France*, 2008. 2.6.3

[56] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR '07, pages 1–10, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 978-1-4244-1749-0. doi: 10.1109/ISMAR.2007.4538852. URL https://doi.org/10.1109/ISMAR.2007.4538852. 2.6.3

[57] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, December 2012. 2.6.1

[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257. 2.1, 2.2.2

[59] Manohar Kuse and Shaojie Shen. Robust camera motion estimation using direct edge alignment and sub-gradient method. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 573–579. IEEE, 2016. 5.2.6

[60] Leaflet. URL https://leafletjs.com/. 3.2.1

[61] MA Lebedev, Yu V Vizilter, OV Vygolov, VA Knyaz, and A Yu Rubis. Change detection in remote sensing images using conditional adversarial networks. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 42(2), 2018. 2.6.2

[62] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-

propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990. 2.1

[63] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 2.1

[64] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 661–670, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2956-9. doi: 10.1145/2623330.2623612. URL http://doi.acm.org/10.1145/2623330.2623612. 2.1

[65] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2359–2367, 2017. 2.2.3

[66] 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 *European conference on computer vision*, pages 740–755. Springer, 2014. 2.2.3, 2.2.5

[67] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 2.2.2

[68] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. 2.1, 4.3, 4.1, 4.6

[69] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1623264.1623280. 5.2.1

[70] S. Maity, A. Saha, and B. Bhowmick. Edge slam: Edge points based monocular visual slam. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 2408–2417, Oct 2017. doi: 10.1109/ICCVW.2017.284. 2.6.3

[71] Google Maps. URL https://www.google.com/maps. 2.5

[72] R. Mur-Artal and J. D. Tards. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, Oct 2017. ISSN 1552-3098. doi: 10.1109/TRO.2017.2705103. 2.6.3, 5.2.6

[73] R. Mur-Artal, J. M. M. Montiel, and J. D. Tards. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, Oct 2015. ISSN 1552-3098. doi: 10.1109/TRO.2015.2463671. 2.6.3, 5.2.6

[74] M. Gdula N. Angelini, J. Brache and G. Shevlin. Gps coordinate pothole mapping. Technical report, Worchester Polytechni, 2006. URL http://users.wpi.edu/~sageman/mqp/docs/executive-summary.doc. 2.6.1

[75] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulò, and Peter Kontschieder. The map-

illary vistas dataset for semantic understanding of street scenes. In *International Conference on Computer Vision (ICCV)*, 2017. URL https://www.mapillary.com/dataset/vistas. 2.2.5

[76] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015. 2.2.1

[77] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1520–1528, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.178. URL http://dx.doi.org/10.1109/ICCV.2015.178. 2.2.1

[78] OpenStreetCam. URL https://openstreetcam.org. 2.5

[79] OpenStreetMap. URL https://www.openstreetmap.org. 2.5

[80] Emanuele Palazzolo and Cyrill Stachniss. Fast image-based geometric change detection given a 3d model. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6308–6315. IEEE, 2018. 2.6.2

[81] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016. 2.2.1

[82] Thomas Pollard and Joseph L Mundy. Change detection in a 3-d world. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–6. IEEE, 2007. 2.6.2

[83] Richard J Radke, Srinivas Andra, Omar Al-Kofahi, and Badrinath Roysam. Image change detection algorithms: a systematic survey. *IEEE transactions on image processing*, 14(3): 294–307, 2005. 4.1

[84] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017. 2.2.2

[85] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 2.2.2

[86] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL http://arxiv.org/abs/1506.01497. 2.3, 2.2.2, 2.2.3, 3.2.3

[87] Roadbotics. URL https://www.roadbotics.com/. 2.6.1

[88] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 2.1, 2.2.1, 4.4

[89] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large

scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 2.1, 2.2.5

[90] Ken Sakurada and Takayuki Okatani. Change detection from a street image pair using cnn features and superpixel segmentation. In *BMVC*, pages 61–1, 2015. 4.1, 4.6

[91] M. Satyanarayanan. Edge computing for situational awareness. In *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–6, June 2017. doi: 10.1109/LANMAN.2017.7972129. 3.1

[92] Mahadev Satyanarayanan, Wei Gao, and Brandon Lucia. The computing landscape of the 21st century. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 45–50. ACM, 2019. 2.6, 2.4

[93] F. Schenk and F. Fraundorfer. Robust edge-based visual odometry using machine-learned edges. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1297–1304, Sep. 2017. doi: 10.1109/IROS.2017.8202305. 2.6.3, 5.2.6, 5.3, 5.3.2

[94] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61: 85–117, 2015. doi: 10.1016/j.neunet.2014.09.003. Published online 2014; based on TR arXiv:1404.7828 [cs.NE]. 2.1

[95] Amazon Web Services. URL `https://aws.amazon.com/`. 2.4

[96] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651, 2017. doi: 10.1109/TPAMI.2016.2572683. URL `https://doi.org/10.1109/TPAMI.2016.2572683`. 2.2.1

[97] Mihail L Sichitiu and Maria Kihl. Inter-vehicle communication systems: a survey. *IEEE Communications Surveys & Tutorials*, 10(2), 2008. 3.1

[98] Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, and Mahadev Satyanarayanan. Scalable crowd-sourcing of video from mobile devices. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 139–152. ACM, 2013. 3.2.3

[99] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2.2.1

[100] Craig Smith. 14 interesting waze statistics and facts (june 2018). URL `https://expandedramblings.com/index.php/waze-statistics-facts/`. 1

[101] F. Steinbrcker, J. Sturm, and D. Cremers. Real-time visual odometry from dense rgb-d images. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 719–722, Nov 2011. doi: 10.1109/ICCVW.2011.6130321. 2.6.3

[102] Simon Stent, Riccardo Gherardi, Björn Stenger, and Roberto Cipolla. Detecting change for multi-view, long-term surface inspection. In *BMVC*, pages 127–1, 2015. 2.6.2

[103] Simon Stent, Riccardo Gherardi, Björn Stenger, Kenichi Soga, and Roberto Cipolla. Visual change detection on tunnel linings. *Machine Vision and Applications*, 27(3):319–330, 2016. 2.6.2

[104] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012. 5.3, 5.3.1, 5.3.1

[105] Teppei Suzuki, Soma Shirakabe, Yudai Miyashita, Akio Nakamura, Yutaka Satoh, and Hirokatsu Kataoka. Semantic change detection with hypermaps. *arXiv preprint arXiv:1604.07513*, 2016. 2.6.2

[106] Christian Szegedy, Scott Reed, Dumitru Erhan, Dragomir Anguelov, and Sergey Ioffe. Scalable, high-quality object detection. *arXiv preprint arXiv:1412.1441*, 2014. 2.2.2

[107] J. J. Tarrio and S. Pedre. Realtime edge-based visual odometry for a monocular camera. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 702–710, Dec 2015. doi: 10.1109/ICCV.2015.87. 2.6.3, 5.3.1

[108] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012. 2.1

[109] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *IEEE transactions on pattern analysis and machine intelligence*, 32(5):815–830, 2009. 4.1, 4.6

[110] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2): 154–171, 2013. 2.2.2

[111] S. Varadharajan, S. Jose, K. Sharma, L. Wander, and C. Mertz. Vision for road inspection. In *IEEE Winter Conference on Applications of Computer Vision*, pages 115–122, March 2014. doi: 10.1109/WACV.2014.6836111. 2.6.1

[112] Ashley Varghese, Jayavardhana Gubbi, Akshaya Ramaswamy, and P Balamuralidhar. Changenet: a deep learning architecture for visual change detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018. 2.6.2

[113] Junjue Wang, Brandon Amos, Anupam Das, Padmanabhan Pillai, Norman Sadeh, and Mahadev Satyanarayanan. A scalable and privacy-aware iot service for live video analytics. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, MMSys'17, pages 38–49, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5002-0. doi: 10.1145/3083187.3083192. URL http://doi.acm.org/10.1145/3083187.3083192. 3.2.3

[114] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. Bandwidth-efficient live video analytics for drones via edge computing. In *SEC '18: Proceedings of the Third ACM/IEEE Symposium on Edge Computing*, New York, NY, USA, 2018. ACM. 3.2.4

[115] Waze. URL https://www.waze.com/. 1, 2.6.1

[116] T. Whelan, M. Kaess, M.F. Fallon, H. Johannsson, J.J. Leonard, and J. McDonald. Kintinuous: Spatially extended kinectfusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia, Jul 2012. 2.6.3

[117] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. *Int. J. Comput. Vision*, 125(1-3):3–18, December 2017. ISSN 0920-5691. doi: 10.1007/s11263-017-1004-z. URL https://doi.org/10.1007/s11263-017-1004-z. 2.6.3

[118] Wang Xin, Wei Dong, Mingcai Zhou, Renju Li, and Hongbin Zha. Edge enhanced direct visual odometry. pages 35.1–35.11, 01 2016. doi: 10.5244/C.30.35. 2.6.3

[119] Yuwen Xiong, Renjie Liao, Hengshuang Zhao, Rui Hu, Min Bai, Ersin Yumer, and Raquel Urtasun. Upsnet: A unified panoptic segmentation network. *arXiv preprint arXiv:1901.03784*, 2019. 2.5, 2.2.4

[120] F. Ye, M. Adams, and S. Roy. V2v wireless communication protocol for rear-end collision avoidance on highways. In *ICC Workshops - 2008 IEEE International Conference on Communications Workshops*, pages 375–379, May 2008. doi: 10.1109/ICCW.2008.77. 3.1