UNIVERSITY OF CALIFORNIA
SANTA CRUZ

TIGHTLY-COUPLED LIDAR AND CAMERA FOR AUTONOMOUS VEHICLES

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL ENGINEERING

by

Mohammad Hossein Daraei

June 2018

The Dissertation of Mohammad Hossein Daraei
is approved:

Professor Hamid Sadjadpour, Chair

Professor Yu Zhang

Professor Roberto Manduchi

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures v

List of Tables vii

Abstract viii

Dedication x

Acknowledgments xi

1 Introduction 1

2 Environment Perception for Autonomous Driving 9
- 2.1 Literature on Environment Perception . . . 10
- 2.2 Measuring the Surroundings . . . 14
- 2.3 Multi-modal Spatio-temporal Association . . . 17
  - 2.3.1 Continuous Mapping Between Sensor Spaces . . . 20
  - 2.3.2 Discrete Multi-modal Measurement Association . . . 22
- 2.4 Distance-aware Processing Resolution . . . 23
- 2.5 Proposed Fusion Architecture . . . 29

3 Velocity Estimation 34
- 3.1 Related Work . . . 36
- 3.2 Problem Formulation . . . 41
- 3.3 Proposed Motion Estimation and Depth Reconstruction . . . 42
  - 3.3.1 Energy Function Derivation . . . 43
  - 3.3.2 LiDAR Data Term $E_{\mathcal{P}}(\mathbf{v}, \mathbf{d})$ . . . 45
  - 3.3.3 Surface Smoothness Term $E_{\mathcal{S}}(\mathbf{d})$ . . . 46
  - 3.3.4 LiDAR Likelihood Term, $E_L(\mathbf{v})$ . . . 49
  - 3.3.5 Image-based Data Term $E_{\mathcal{I}}(\mathbf{v})$ . . . 50
  - 3.3.6 Occlusion and parallax modeling . . . 53
- 3.4 Optimization . . . 54

3.4.1 Inference and Tracking 54
3.4.2 Non-linear Penalization 57
3.4.3 Multi-resolution analysis 58
3.5 B-Spline Wavelets as Basis Functions 59
3.6 Experiments 61
3.6.1 Hardware setup discussion 61
3.6.2 Sensor Calibration 63
3.6.3 Evaluation metrics 65
3.6.4 Velocity Estimation w/o Tracking 65
3.6.5 Velocity Estimation + Tracking 70

4 Measurement Grouping **77**
4.1 Prior Work 79
4.2 Problem Statement 81
4.3 Proposed Method 82
4.3.1 Preprocessing 83
4.3.2 Segment initialization 83
4.3.3 Surface reconstruction 85
4.3.4 MRF-based mask optimization 86
4.3.5 First-order potentials 88
4.3.6 Pairwise potentials 88
4.4 Experimental Results 89
4.4.1 Implementation 90
4.4.2 Discussion 92
4.4.3 Parallax and class-independence 93

5 GPU Implementation **94**
5.1 Practical Adaptations 95
5.2 CUDA Kernels 97
5.2.1 Plane-level kernels 98
5.2.2 Tile-level kernels 101
5.2.3 Inter-tile kernels 104

6 Conclusion **107**

Bibliography **111**

List of Figures

2.1 Sensor positioning, reference systems, and data structures for LiDAR point clouds (left) and images (right). All pixels in an image are captured at the same time, while LiDAR measurements have different timestamps depending on their azimuth φ. 18
2.3 An example of parallax, caused by the distance between sensors. In this case, LiDAR is mounted on front bumper, and Camera is mounted behind the windshield. 22
2.5 A representation of how intermediate surface $\mathcal{F}$ is employed to both convert image-plane displacements to measurements on velocity and facilitating point-set registration. Red ellipsoids represent the uncertainty in measurements. 24
2.8 High-level block diagram of our environment perception module . 30

3.1 Our tracking and reconstruction (a) takes sequences of LiDAR point clouds and camera images, and (b) finds 3D velocity $\mathbf{v}$ for each object. It (c) estimates an intermediate surface $\mathcal{F}$ to fuse sensors, and (d) could potentially accumulates points as a 3D model. 35
3.3 Block-diagram for the proposed velocity and surface estimation method. Objects are modeled with surface $\mathcal{F}$, appearance $\mathcal{M}$, and velocity $\mathbf{v}$. 43

3.4 Reconstructed surface $\mathcal{F}$ for a vehicle (a) projected on the image, (b) represented as a 2D depth map on discrete $\theta\varphi$ grid, and (c) corresponding weights for each grid cell. The surface is allowed to bend along patterns with low weights. 49
3.13 Ground truth and estimated pitch angle, θ_g and $\hat{\theta}$, for different methods over the entire sequence 68

4.1 Given (a) an image $\mathcal{I}$ and a point cloud $\mathcal{P}$ and a point p_s, we compute (b) MRF unary potentials based on LiDAR points, and (c) MRF pairwise weights from image gradients on a 2D angular grid $\Omega_\epsilon = (\theta, \phi)$. The MRF solution is (d) a segment covering LiDAR points while aware of image edges. The final output is (e) a three-dimensional surface segment (or a supersurface). 79
4.3 Each segment s has (a) a set of assigned points (in green) and a square box $\mathcal{B}_s$ centered on its anchor p_s, then (b) a 3D surface $\mathcal{F}_s$ is fit to assigned points. 84
4.5 Superpixels generated for (a) a sample scene using different methods, (b) proposed method, (c) SLIC [6], (d) DASP [113]. No superpixel is generated for the pink region. 90
4.6 Transferring supersurfaces from Ω_ϵ to image plane is susceptible to parallax, as the pole occludes the car in (a). This can be solved through (b) depth-ordering of surface segments. 90

5.1 Schematic representation of fusion planes in the multi-resolution pyramid, image plane and depth-map at each level, grid of structured overlapping tiles, and tile-level variables in each grid. 97
5.7 Examples of object masks, with hue encoding direction and saturation encoding magnitude of estimated velocity with respect to ground. 106

List of Tables

1.1 A brief chronology of influential events in the history of autonomous driving [3]. 5

3.1 Comparison of four LiDAR sensors for automotive applications . . 62
3.2 Magnitude of velocity error ($\|\Delta\mathbf{v}\|$) and crispness score for different methods tested on our collected data. 67
3.3 Results of different methods tested on KITTI Raw [36] 71

4.1 Average boundary recall (Avg BR) and average number of segments for non-ground regions (Avg Seg) for two image-based (SLIC, Veksler), one RGB-based (DASP), and the proposed method. 93

Abstract

Tightly-Coupled LiDAR and Camera for Autonomous Vehicles

by

Mohammad Hossein Daraei

Autonomous driving has received remarkable attention and investment from both industry and academia over the past couple of years. Researchers from different fields are contributing to solve various challenges towards fully autonomous cars, e.g. optics, light sensing, deep learning, statistics. A driver-less car, first and foremost, must be equipped with the right choice of complimentary sensors in order to measure the surroundings. Then it needs to process the measurements to understand the scene, perform path planning, and react to the environment. In this work, we focus on processing raw sensor measurements to extract meaningful physical or geometrical features like segments, velocity, depth, graphical relations between different segments, etc.

In this dissertation, our primary focus is on leveraging the complimentary features of two automotive grade sensors: LiDAR and Camera. The goal is to break the scene into independently moving segments, reconstruct their surface, find inter-segment connections, and estimate their three-dimensional velocity. As the main contribution of this work, we combine measurements in LiDAR points and Camera pixels at the raw level. This is opposed to tracking-by-detection or high-level fusion where sensors are fused after object detection. We lay a graph-based approach to generate a set of three-dimensional superpixels (supersurfaces) that cover the whole environment. These supersurfaces are optimized according to connectivity features in both LiDAR and Camera space. At a higher level, the connections between neighboring supersurfaces are also modeled and considered in

subsequent modules. We also develop an energy function that explicitly penalizes three-dimensional velocity vectors based on both image pixels (optical flow term) and LiDAR points (geometric distance term). This energy is further regularized to maintain edge-preserving temporal and spatial smoothness. This leads to tightly-coupled sensor fusion, and benefits from more robustness in velocity estimation.

One important feature of this work is its independence from object detection, unlike many methods that rely on supervised learning to detect object instances. This feature makes our algorithm capable of handling unknown classes of objects. Another key characteristic of our fusion approach is the notion of adaptive *processing resolution*. Objects, depending on their distance, are processed at different pyramid levels (and different resolutions) which eliminates redundant computations and still maintains desired estimation error bounds.

Our prototyped experimental results show an improvement in velocity estimation error with respect to the state of the art. We also demonstrate the functionality of the overall algorithm by assembling all submodules into an efficient CUDA implementation. Several changes and adaptations are applied to make the algorithm optimized for running on GPU.

Dedicated to Shahab al-Din Abu Hafs Umar Suhrawardi (1154-1191)

Acknowledgments

I want to express my gratitude to all the individuals who contributed to this research, those who kindly supported me, and the ones who inspired me and helped flicker new insights.

Most importantly, I would like to thank my PhD supervisor Roberto Manduchi. He guided me through challenges of this research, and always provided me with eye-opening insights and wise comments. I also appreciate other members of Computer Vision Lab [1] of UC Santa Cruz for insightful discussions and exchange of ideas, especially Seongdo Kim, Siyan Qin, and Brigit Schroeder.

This research would not have been possible without the generous support of Volkswagen Group of America's Electronic Research Lab (ERL) [2]. In particular, I would like to thank Jörg Schlinkheider, former head of Driver Assistance Systems, Nikhil George, head of Perception and Machine Learning, and Lutz Junge, Program Manager at ERL. They continually trusted in this research direction, and made the results possible. I also appreciate the support and help of my co-author from ERL, Anh Vu, who provided me with great insights. With his expertise in the area, knowledge of the literature, and our long technical discussions he effectively served as my second supervisor. I also thank my colleagues and friends at ERL for their comments and ideas: Kaushik Raghu, Jerramy Gipson, Nils Kuepper, and Edmund Zink.

I finally thank my parents and my sister, Sara, whose unconditional love and support always brightened the way, and helped me in the moments of frustration.

[1] `http://vision.soe.ucsc.edu`
[2] `http://www.vwerl.com/`

Chapter 1

Introduction

Over the past two decades, Autonomous Vehicles (AV) and Driver Assistance Systems (DAS) have gained remarkable attention and have been extensively studied. It's undeniable that these technologies will drastically reduce the number of car accidents and fatalities in the future. Although first experiments with self-driving cars date back to early 50s, it was until 90s that major breakthroughs took place. Since *DARPA Grand Challenge (2004, 2005)* and *Urban Challenge (2007)* [103, 16], many researchers have become interested in this area, as well as huge companies in tech and auto industry. Waymo, Tesla Motors, Volkswagen Group, Mercedes Benz, GM Cruise, Ford, and NVIDIA are among the growing list of companies who are currently testing their autonomous vehicles on California roads, according to disengagement reports [1] submitted to Department of Motor Vehicles (DMV). Table 1.1 summarizes important events in the history of autonomous driving. These events have led to commercialized DAS features, which has made Levels 2-3 of *SAE International Levels of Driving Automation* [1] realistic. DAS aims at providing assistance features such as lane centering, adaptive cruise control (ACC), automatic steering, parallel parking, etc. These features

[1] `http://www.sae.org/misc/pdfs/automated_driving.pdf`

assist the driver with driving, while they are still in charge and required to take over when alarmed about an upcoming disengagement. Most of these features are now available as options for cars in production, e.g. Audi Q7 [90], Mercedes-Benz S-Class.

Fully autonomous vehicles, *SAE automation levels 4-5*, have a much more complicated mission and yet not fully available. They must be extremely reliable, safe, and robust to unseen phenomenon given their highly sensitive role. A fully autonomous vehicle must execute all operations, monitor itself, and be able to handle all unprecedented events and conditions, e.g. roads without markings, unexpected objects and debris on the road, unseen environments, adverse weather, etc. Although several systems, like Tesla Autopilot, Waymo, and Uber cars, are seemingly able to drive autonomously on highways and most urban environments, they are far from exhibiting level 5 or even level 4 autonomy. There are myriads of examples of complicated scenarios (corner cases) in which these systems disengage and ask the driver to take over. This is evident especially in the legal arguments that follow after some of the fatality accidents involving these cars. This calls for better sensors, better processing power, and better algorithm before fully-autonomous cars are reality.

One of the ongoing debates among driver-less car designers is about the choice of perception sensors and their mounting position. In Sec. 2.2 we study automotive grade sensors that are typically deployed for driver-less cars and their important characteristics. One of these sensors is LiDAR, which actively measures range on an azimuth-elevation grid using multiple laserscanners and a rotating mirror. LiDAR has been a challenging sensor for mass production, mainly due to its relative high price. Some companies like Tesla, Comma.ai, and AutoX have pushed the idea that, similar to humans, cars can perceive and navigate using just

the eyes. Some of them are even striving towards an end-to-end pure image-based neural network solution to autonomous driving. But LiDAR technology is growing with remarkable investments [2], and sensor prices are dramatically dropping [5]. Thus most companies are planning to actually make use of multiple LiDARs. Most autonomous car manufacturers are designing multi-module systems to model the environment and make decisions. As a result, fusing the sensors through geometric and information-theoretic methods is inevitable. Most of the research on sensor fusion has been devoted to studying how to combine detected objects in each sensor space. The philosophy behind this methodology models the world as a set of three-dimensional bounding boxes encompassing objects of different types. This approach is highly dependent on object classifiers, which are trained in a supervised manner to detect instances of a particular object. These object lists are further associated and processed along time and generate object tracks and facilitate velocity estimation. This is what we refer to as *loosely-coupled* or *high-level* fusion of the two sensors.

The majority of the literature on environment perception for autonomous driving has been devoted to object classification or semantic segmentation [9, 39, 38] or reconstruction [97]. This is partly due to recent developments in supervised learning and deep neural networks. Improvements in classification and localization leads to more reliable high-level fusion and tracking-by-detection. Some of these networks have been efficiently implemented on GPUs – achieving real-time performance. With the introduction of more advanced GPUs, convolutional neural networks are now employed for even more complicated tasks, e.g. segmentation [9], optical flow [53], tracking [75], localization [83, 84]. This calls for more data, efficient data collection, easier labeling procedure, less sensitivity to dataset size, and even the use of simulated datasets in conjunction with domain adaptation.

Generalization error may also become an issue if over-fitting or under-fitting is present in the dataset. For example, a network trained with data from a specific city and collected by a specific sensor might not be applicable to other conditions. Although there are learning-based solutions to movement detection and optical flow estimation [92, 53, 75], we limit the use of machine learning to tuning hyper-parameters of our probabilistic models. This is mainly because we intend to preserve the universality of algorithms focus on statistical fusion.

In this study, we turn our attention to a learning-free approach for sensor fusion. This approach [22, 21] does not require a list of detected objects, rather performs fusion on raw sensor measurements, referred to as *tightly-coupled* or *low-level* fusion. This helps achieve more robust and more accurate velocity estimation, and doesn't require or rely on object detection. In addition, it can deal with unknown object categories and has the capability of handling objects at various scales to compromise between performance and accuracy. The main goal of this research is to combine the complimentary features of LiDARs and Cameras in a Bayesian framework, and to find a robust method that locates and tracks objects independent of their type – leveraging the information coming from each sensor modality. This is equivalent to a scene reconstruction algorithm that populates each segment (or cell) with a three-dimensional velocity vector. Using robust probabilistic formulation of the problem, each estimated variable also comes with uncertainty bounds.

In Chap. 2, we study the characteristics of different sensors, identify the challenges that exist in the way of fusing them, and present a high-level depiction of our proposed system. A brief discussion on the nature of LiDAR and Camera measurements and their estimation-theoretical implications is also provided. We investigate the geometrical relationships between the measurements from two dif-

Table 1.1: A brief chronology of influential events in the history of autonomous driving [3].

1980s	A vision-guided Mercedes-Benz robotic van achieves a speed of 39 mph on streets without traffic
	Carnegie Mellon University pioneers the use of neural networks for controlling vehicles
1994	VaMP of Daimler-Benz drove 620 miles on a highway with human interventions
1995	Carnegie Mellon Navlab's semi-autonomous car completes 3,100 miles (autonomous steering)
	A re-engineered autonomous S-Class Mercedes-Benz completes a 990 mile European journey
1996	University of Parma's stereo-vision-enabled ARGO project completes 1,200 miles on highways
1998	Toyota the first to introduces laser-based Adaptive Cruise Control (ACC)
2004	**DARPA Grand Challenge**: 150 miles in desert. No winner.
2005	**DARPA Grand Challenge II**: in a desert environment with maps. 5 winners.
	BMW starts working on autonomous driving
2007	**DARPA Grand Challenge III (Urban Challenge)**: Carnegie Mellon University wins.
2009	Google begins developing its self-driving car
2010	Audi sends a driverless TTS to the top of Pike's Peak at race speeds
	University of Parma's VisLab conducts the first intercontinental autonomous challenge
	Universitat Braunschweig's vehicle the first car licensed for autonomous testing in Germany
2011	GM created the EN-V (Electric Networked Vehicle), an autonomous urban car
	"Spirit of Berlin" and "MadeInGermany" are tested to handle traffic, traffic lights, and roundabouts
2012	Volkswagen begins testing Autopilot: with a speed up to 80mph on highways
	Google self-driving car conducts a 14-mile driving test in Las Vegas, Nevada
2013	VisLab conducts a successful autonomous urban test, with no human control
	Daimler R&D's S-class drives autonomously for 100km, using stereo-vision and radars
	Nissan Leaf with semi-autonomous features granted license to drive on Japanese highways
	Mercedes S-class has options for autonomous steering, lane keeping, parking, etc.
2014	Navia shuttle, limited to 12.5mph, becomes the first self-driving vehicle for commercial sale
2015	Volvo announces its plans to work on autonomous driving
	Tesla Motors introduces AutoPilot through a software update
	Uber announces partnership with Carnegie Mellon University to develop autonomous cars
	Delphi Automotive completes the first automated coast-to-coast journey in the US
2016	The first fatal accident involving a Tesla AutoPilot in Florida, raising legal concerns
	Ford Motor Company announces partnership with Velodyne LiDAR for its next generation R& D cars
	Singapore launches the first self-driving taxi service: nuTonomy
2017	Apple is reportedly doing research on 3d laserscanners for self-driving cars
	Velodyne announces VLS-128, its best laserscanner with 300m range and 128 vertical layers
	Waymo finishes 2 million miles of autonomous driving
	Audi A8 marked as first manufacture car to reach level-3 autonomous driving
2018	Waymo announces partnership with Jaguar Land Rover
	Uber causing a fatality involving autonomous vehicles on public roads

ferent modalities, the characteristics of each sensor space, and how measurements can be associated spatially and temporally. As another contribution, we study the role of *processing resolution* in order to analyze objects at different depths. We argue that, in order to achieve a desired velocity estimation accuracy ϵ, objects at different distances need to be processed at different resolutions. This is equivalent to constructing a Gaussian pyramid (with down-sampling factor of 2) and processing different objects at different pyramid levels. In particular, we argue processing closer objects at a low-resolution level saves tremendous computation effort while still achieving the desired error bound ϵ.

Chap. 3 is devoted entirely to our fusion-based velocity estimation method [22]. Throughout this chapter, scene segmentation or measurement grouping is assumed to be provided. We specifically focus on how to combine what all pixels and points in an object can tell about its motion. We formulate the problem as Bayesian estimation of the latent variable, three-dimensional velocity $\mathbf{v}$, which includes LiDAR-based and Camera-based likelihood functions. Then we proceed to minimize the energy form of this probabilistic model. This is equivalent to maximizing the temporal measurement matching in both sensor spaces. If objects are non-deformable and brightness constancy assumption is not severely violated, matching of consecutive measurements essentially provides object velocity. The LiDAR term penalizes velocity according to the geometric distance of current and previous points. And the Camera term minimizes the photometric distance of current and previous image patches. But the Camera term (similar to optical flow computation) results in two dimensional displacement vectors on the image plane. It is possible to convert optical flow measurements into direct measurements of three-dimensional velocity if per pixel depth is available [87]. In most cases, however, projections of LiDAR points only cover a small fraction of pixels. One

of the main contributions of this work is the introduction of an intermediate surface $\mathcal{F}$. It provides dense depth for all pixels and simplifies LiDAR geometric matching as well. We conduct several experiments using different LiDARs (Valeo Ibeo ScaLa B2 and Velodyne HDL-64E) to demonstrate the superiority of our method compared to state of the art velocity estimation.

The proposed measurement grouping algorithm (published in [21]) is explained in Chap. 4. The goal is to group raw pixels and points into coherent segments (referred to as supersurfaces) that can be assembled at a higher level into objects. This is similar to superpixel segmentation of an image, while supersurfaces are three-dimensional and come with an underlying surface model. We aim to generate distance-aware segments that have a constant metric size. As a result, farther supersurfaces appear smaller on image domain compared to closer ones. We formulate the problem as a multi-label Markov Random Field where first order potentials come from LiDAR points and pairwise potentials are based on image edges. It is shown that supersurfaces are more efficient than other over-segmentation methods, since we need less supersurfaces on average to retain the same boundary recall. A move-making algorithm like α-expansion is necessary for multi-label MRF inference, which results in expensive computation. In our CUDA implementation, the MRF model is replaced with a Diffusion process and solved by iterative filtering. It turns out this is a very efficient approximation and suitable for data parallelism on GPU. Although not the main focus of this research, we also lay a geometric scoring function to determine whether two nearby supersurfaces are part of the same rigid object. Measurement grouping and variable estimation are sometimes solved jointly. In many perception modules, however, measurement grouping and velocity estimation are served as separate problems. There are some methods [116] that refine superpixels or segments after updating

velocity, and continue alternating between these two steps. We take a similar approach and refine measurement grouping after velocity estimation. We down-weight those measurements that exhibit large matching error when translated by the estimated segment velocity.

In Chap. 5 we focus on assembling grouping and estimation into one coherent system that can achieve real-time performance. The discussion in this chapter is deliberately more practical, and we simplify and in some cases ignore some of the ideas developed in the previous chapters. For example, we perform depth extrapolation instead of explicit surface reconstruction. We also replace the irregular supersurfaces of Chap. 4 with an ordered grid of overlapping tiles. Some of these changes are applied due to GPU architecture specifications, like limited shared memory in a thread block, or maximum number of threads per block. We also decrease the number of bit each variable is encoded with, so that readings from GPU global memory are minimized.

Chap. 6 concludes the dissertation by reviewing the key features of the proposed method and our main contribution. We also put forth potential directions for building upon the algorithms developed in this thesis. In particular, we list some of the modules in the proposed architecture that would benefit from supervised learning.

Chapter 2

Environment Perception for Autonomous Driving

Autonomous driving research entails a broad spectrum of fields: control theory, computer vision, machine learning, optics and sensing, statistics, human-machine interaction, etc. Several real-time components are required in order to make a vehicle drive autonomously and safely, including environment perception, ego-vehicle localization, path planning, and control [62]. Scene understanding is an essential component for an autonomous vehicle to safely navigate through dynamic environments. In this dissertation, most of the attention is on scene understanding while remaining agnostic about its semantic content and object classes. This approach has the advantage of being immune from object detection errors, and it also is capable of handling unseen obstacles.

In order to model and analyze the surroundings, the vehicle must segment the scene into moving objects, reconstruct their shape, and find their 3D motion. For this purpose, sensor fusion plays an important role in extracting relevant information from different sensors with different characteristics. We argue that sensors need to be fused in a *tightly-coupled* fashion, in order to gain more ro-

bustness. This is equivalent to fusing sensor spaces at raw measurement level, and generating joint features and models at such low level. This is opposite to loosely-coupled (high-level) fusion, in which features, models, and even tracks are generated separately and merged at object level. A tightly-coupled environment perception module must jointly processes raw sensor measurements and build up a rich representation of the surrounding. Measurement in this context means any LiDAR point or Camera pixel. Such environment perception module must then provide,

1. Grouping of measurements into clusters
2. Assignment weights connecting each measurement to each cluster
3. Estimated 3D velocity vector for each cluster
4. Reconstructed dense surface for each cluster

After a brief glance at the environment perception literature, we review available automotive sensors, study their strengths and shortcomings, and explain the reasoning behind a LiDAR-Camera combination. Then an overview of current trends in perception research for autonomous driving is portrayed. Finally the problem of tightly-coupling LiDAR and Camera is explained, its geometrical constraints and rate-resolution characteristics are studied, and the main challenges and obstacles are listed.

2.1 Literature on Environment Perception

While we do not study all necessary components in an autonomous vehicle, it is important to have an understanding of their function.

Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is a critical task which determines the position and velocity of ego-vehicle relative to the world reference or a map. This problem is typically solved using one or a combination of sensors, such as Global Positioning System (GPS) and Inertial Measurement Units (IMU). Many researchers have also employed visual and depth information in conjunction with available maps of the environment to improve localization. When visual or morphological maps are available, the autonomous vehicle can localize itself by matching sensor features to maps. This is typically referred to as Visual Odometry [24, 47, 57]. For localization based on images, it is important to only extract and match features that belong to static sections of the environment. Presence of unidentified dynamic objects has the potential to corrupt ego-motion estimation using camera.

Semantic Segmentation

Another essential piece of knowledge is the semantic content of the scene. Semantic segmentation is the problem of finding the underlying class (or label) of each measurement: whether it belongs to a car, a pedestrian, the road, etc. The significance of semantic information is twofold. First, the vehicle must behave differently with different object classes. Also, object class (or any extracted semantic feature) could help with segmentation and velocity estimation, in the least through providing class-specific velocity and shape priors. Semantic segmentation and has been studied extensively, especially recently and in the context of Deep Learning. Several architectures have been proposed to semantically segment images, e.g. Fully Convolutional Networks (FCN) [65], Segnet [9], CRF-CNN [121]. Instance-level segmentation is a similar problem that identifies instances of a par-

ticular object class. Several trainable models have been proposed in the past decades, e.g. Gaussian Mixture Models (GMM) [79], Support Vector Machines (SVM) on Histogram of Oriented Gradients (HOG) [33], and Deformable Parts Model (DPM) [28, 77, 18]. But they are all outperformed by more recent Deep Learning architectures, e.g. R-CNN [39, 38], YOLO [91], VoxelNet [122]. These networks generate candidates as 2D bounding box, 3D cuboids, or even nonparametric masks. Convolutional networks require a dense data grid as input, thus they are directly applicable to camera images. However, after appropriate preprocessing or employing convolution components capable of processing sparse data, it is possible to use them on LiDAR point clouds and extract 3D objects [67, 122]. Instance-level detection (or segmentation) generates a list of objects along with their position and class. Having such list naturally encourages us to perform frame-to-frame association, and thus estimate velocity and create object tracks. This is typically referred to as Tracking by Detection. While being a conventional high-level motion estimation algorithm, it directly depends on reliable detection and works for known classes of objects for which trained models are available.

High-level Tracking by Detection

In order to both detect and track objects of interest, some of the work on fusing lidar and camera generates a region of interest (ROI) based on detected candidates (hypothesis generation) in lidar space, and project it to the image space for checking and tracking (hypothesis verification), e.g. [79, 34, 33, 41, 19, 51]. The task of tracking, in most of these techniques, is mainly based on the primary sensor (LiDAR), and the secondary sensor (camera) serves as an auxiliary source to confirm candidates. Nuss et al, [74] first apply a trained neural network to classify each pixel as vehicle, road, or static. They project labeled pixels onto the

lidar space, and merge the sensor data in an occupancy grid. Several methods, e.g. [73, 50, 109], treat the sensors independently, generate separate candidates in both image and lidar spaces, and then try to associate detections and track each object.

Low-level Class-independent Motion Estimation

This is the primary domain of this dissertation. A low-level fusion algorithm aggregates raw sensor measurements at an early stage to achieve more reliable features. In computer vision, this group of problems are often referred to as early vision problems. Examples are dense disparity estimation from stereo pair, optical flow estimation, and scene flow [106] estimation. Optical flow is typically referred to the displacement vector (in pixels, or pixels per second) that matches each pixel to the next frame. It does not describe the motion normal to image plane. Scene flow, however, is referred to the joint estimation of pixel depth, motion component in image plane, and motion component normal to image plane.

With a single camera the problem of computing scene flow is highly under-determined. If the camera is not moving we cannot say much about depth; even if it is, the ambiguity lies in the distinction between camera motion and object motion [111]. Most scene flow methods employ a multi-view camera system [108, 69, 116], some have deployed RGBD sensors [87] to directly measure dense depth, and others have made use of LiDARs [52] to get sparse depth. Since optical flow is the projection of the 3D motion field on the camera image plane, an intuitive way to compute scene flow is to reconstruct it from the estimated optical flow [106]. However if the scene structure is not known, it would be difficult to recover scene flow under inconsistent optical flow fields [87]. While [111] argues decoupling of depth and velocity enables us to choose an arbitrary disparity technique, it ignores

exploiting the spatio-temporal information [87]. Thus one prominent line of work is to treat optical flow and disparity as joint measurements and aggregate them in specific constant-motion units. Some examples of these units are shown in Fig. 2.9.

2.2 Measuring the Surroundings

Due to highly dynamic nature of driving scenes, an autonomous vehicle must continually read measurements of its environment. There must be meaningful spatial and temporal connections between all readings so that they can be interpreted in a coherent space-time reference frame. The measurements must then be provided by sensors that are calibrated and synchronized. Typical sensors are monocular or stereoscopic cameras, radar, multi-layer range-finders, 3D LiDARs, ultrasonic sensors, etc. These devices measure different wave sources and sense different physical phenomenon. They have specific characteristics that make them suitable for particular tasks under specific conditions. In this section, we briefly study the strengths and weaknesses of each sensor and point out some of their complimentary features.

Camera

Most driving functionalities rely heavily on receiving and processing signals in the visible light spectrum. All driverless cars benefit from a camera-based setup, e.g. monocular vision [18, 56, 77, 80, 115], stereo vision [105, 26]. Inexpensive cost, high resolution (as opposed to conventional LiDAR), color perception, and rich semantic information are among their advantages. They are by far the most employed sensor for detecting vehicles, pedestrians, road markings, and traffic signs. 3D localization and tracking, however, require depth information which

would only be available with a multi-camera system. Even in a stereoscopic vision system, the estimated depth accuracies are data-dependent and lower than using active range-finders. Cameras are, in general, sensitive to lighting and weather conditions.

Ultrasonic Transducers

Ultrasound sensors are capable of converting a sound wave into electric current, and vice versa. They typically have a frequency range of 40 kHz - 70 kHz for automotive applications. Use case examples are ultrasonic park assist, self-parking, blind spot detection, and valet parking. They are not reliable sensors for measuring position and velocity of remote targets, nor do they provide a dense measurement grid. We do not employ ultrasonic sensors in our proposed setup.

Radar

Automotive millimeter wave radar technology has been employed for Adaptive Cruise Control (ACC), collision warning, stop-and-go, and blind spot detection. Radar transmitter emits radio signals, and measures Doppler frequency shift to detect targets and their relative velocity. Long-range radars are suitable for a maximum range of 200m with frequencies around $76-77$ GHz. Short/mid-range radar, on the other hand, has a wider field of view (greater than $30°$), operates in $24-29$ GHz or $77-81$ GHz, and is a suitable choice for objects in $30-50$m [40]. In general, by going to higher frequencies angular resolution can be enhanced, resulting in a higher speed resolution. Range resolution, however, depends on the modulated signal bandwidth. Unlike LiDAR and camera, mm-wave radars can operate well in severe weather conditions, e.g. snow, fog, and dirt. They can even see physically occluded objects and measure their relative speed. A big

disadvantage of radars is their limited resolution which is a result of regulations that limit their bandwidth. They have a hard time accurately localizing targets, and cannot provide dense depth measurements as LiDARs do. They provide an accurate high-level representation of the scene [66], but cannot say much about lower-level features such as object structures.

LiDAR

Light Detection and Ranging (LiDAR) is a light-based ranging system which transmits $600 - 1000$nm invisible laser. It includes a *rotating mirror* to reflect modulated beams at various angles. Reflections from the environment are sensed using a photodetector, and range values are estimated based on time-of-flight or phase-shift. The result is a semi-dense 3D point cloud of the environment. Reflection echo-width and amplitude are also measured for each pulse. Since LiDARs use invisible light, they don't interfere with ambient light and work equally well under different lighting conditions. LiDARs can *directly and very quickly measure the presence of obstacles along with their position.* This feature alone makes LiDARs a popular and useful sensor for environment understanding [46, 110, 25, 78]. LiDARs provide accurate and direct structure measurements on scene geometry. Unlike stereo systems, these measurements come with no post-processing cost and do not rely on feature-matching algorithms. They, however, have a maximum sensing range of $70 - 100$m, relatively low refresh rates, and sensing problems in adverse weather like rain, snow, fog, or dust. One other issue with LiDAR is presence of dark or low-reflective obstacles. Several methods have been proposed in the rest of this dissertation to remain robust to such outlier or missing data issues.

There has been remarkable effort from the industry side over the past couple

of years to improve LiDAR technology and make it more affordable. This is in part due to the demand for more reliable active sensors from the auto industry. Companies like Waymo, Velodyne, Strobe (acquired by GM), Quanergy, Innoviz, and Luminar are striving towards highly efficient LiDARs with higher resolution and frame rate. Velodyne [1] has recently unveiled their VLS-128 with a maximum range of $300m$. Luminar, for example, has brought the cost of its receivers down from tens of thousands of dollars per unit to just $ 3. [2] They are also capable of better handling dark or unreflective objects. With these developments, the role of LiDAR as a major sensor in the sensor stack of future autonomous vehicles is undeniable. The methods proposed in this dissertation do not make any assumptions about the model or specifications of the hardware deployed, and we experiment with different hardware to prove the compatibility of our fusion approach. In fact, experimental results with two different laserscanners (4-layer and 64-layer) shown in Chap. 3 justify the applicability of the proposed method to different devices.

2.3 Multi-modal Spatio-temporal Association

This section is devoted to the analysis of joint LiDAR-Camera measurement space and its geometric characteristics. We lay spatio-temporal considerations to balance performance versus accuracy from an estimation theory perspective.

Camera measurements are dense grids of intensity values, resulting from Charged-coupled Device (CCD) readings. CCDs are large arrays of pixels, made from p-doped metal-oxide-semiconductors. The dimensions of each pixel (camera resolution) limits the size of observable visual features. These capacitors are activated during image acquisition, convert incoming photons into electric charge, and shift

[1] 128 Lasers on the Car Go Round and Round: David Hall on Velodyne's New Sensor

[2] LIDAR maker Luminar is scaling up and slashing costs in effort to dominate self-driving cars, Andrew J. Hawkins, The Verge

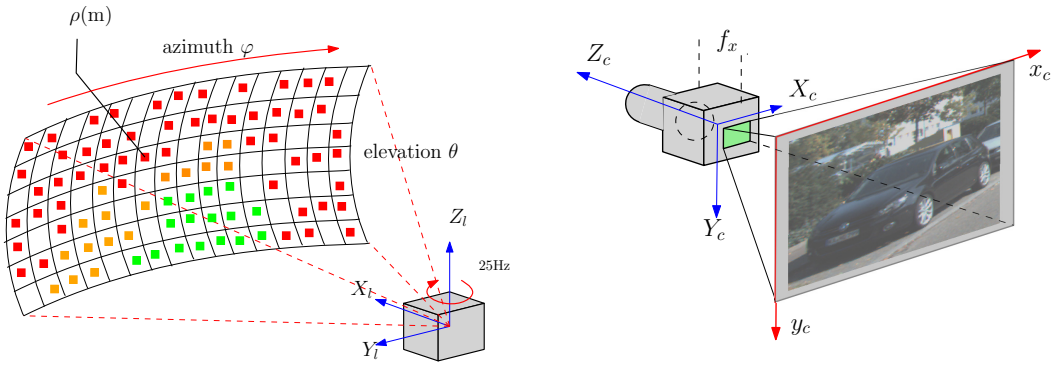


Figure 2.1: Sensor positioning, reference systems, and data structures for LiDAR point clouds (left) and images (right). All pixels in an image are captured at the same time, while LiDAR measurements have different timestamps depending on their azimuth φ.

it into capacitive bins. As a result, the entire image is captured at once and all pixels have roughly the same timestamp. Because of the spinning mirror in most LiDARs, they are essentially measuring range for a dense azimuth-elevation pattern. This pattern has a fixed angular resolution for each axis, depending on the number of vertical layers (laserscanners) and the number of sampling points per cycle. This is shown in Fig. 2.1. Similar to cameras and as a result of diverging angular rays, we get fewer samples (per unit area) for farther objects. In many applications LiDAR measurements are regarded as sparse clouds of points in a Cartesian coordinate system. This perspective, however, ignores the neighborhood relationships between range measurements on the azimuth-elevation grid, shown in Fig. 2.1. As a result of the rotating laserscanners, LiDAR range values are not measured at the same time and have different timestamps. This is critical if the sensor (mounted on ego-vehicle) is moving with respect to the scene, and leads to a drift in the point cloud. This drift can be removed if ego-motion is measured by an IMU, as done in KITTI dataset [36]. There is another type of drift in the point clouds induced by moving objects in the scene. To undo this

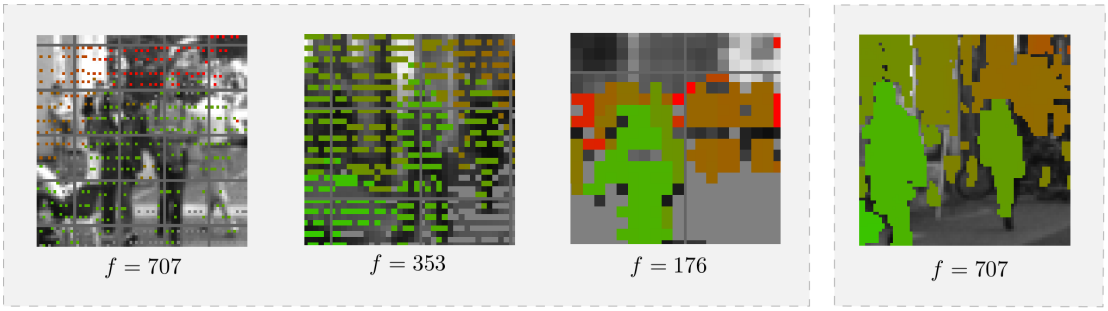


Figure 2.2: Projection of LiDAR point cloud onto images with different focal lengths (left box) and an example of surface reconstruction (right). If camera focal length (hence relative image resolution) is too large then most pixels will not be associated with a LiDAR range measurement.

type of drift is linked with dynamic object velocity estimation problem, and could be done in a joint fashion.

One fundamental necessity of low-level sensor fusion is to associate and group different sensor measurements in space and time. Thus all sensors must be synchronized and perfectly calibrated to vehicle reference system $\mathcal{V}$. This is to ensure their continuous spatio-temporal information are meaningful relative to a unified space-time reference. But this continuous space-time transformation is not enough to associate raw measurements due to their *discrete sampling* nature. LiDAR and Camera have potentially different resolutions and sampling rates. When projected onto a higher resolution image, LiDAR points do not necessarily cover all pixels. Fig. 2.2 (left) shows projections of a LiDAR point cloud onto images of different focal lengths. When image resolution is too low then each pixel will be assigned a depth measurement. But this is not usually the case. The quantitative analysis in Eq. 2.7 suggests the distance between nearby projections is roughly equal to $f\Delta\varphi$, with f denoting camera focal length and $\Delta\varphi$ being the angular distance (in radians) between adjacent LiDAR rays. In order for all image pixels to be covered by LiDAR point projections, $f\Delta\varphi$ must then be smaller than 1. For a typical camera with $f = 1000$, we thus need $\Delta\varphi$ to be smaller than 0.057°. Point

clouds in KITTI dataset, for example, are collected using Velodyne HDL-64E with $\Delta\varphi \approx 0.4°$. Velodyne's recently unveiled VLS-128 has $\Delta\varphi \approx 0.1°$, and Luminar's LiDAR is believed to have $\Delta\varphi \approx 0.05°$ [4].

Until Luminar's LiDAR is made available, a dense depth reconstruction method is then inevitable to extrapolate depth for all pixels. An example is shown in Fig. 2.2 (right). Also some LiDARs have the option of balancing between frame rate and resolution and even foveation functionality. Therefore we need a general criteria for coupling LiDAR and Camera measurements (points and pixels) across time and space.

2.3.1 Continuous Mapping Between Sensor Spaces

In order to associate pixel coordinates with three-dimensional rays in the physical environment, a 3×4 upper-triangular camera calibration matrix $\mathbf{K}_{\mathcal{C}}$ is needed. This is referred to as camera projection matrix, and maps inhomogeneous points in 3D camera reference system onto homogeneous 2D image. In a typical pinhole model, camera projection matrix has 4 degrees of freedom known as *intrinsic parameters*. It is formulated as,

$$\mathbf{K}_{\mathcal{C}} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

where f_x and f_y are respectively horizontal and vertical focal length in pixels, and (c_x, c_y) is camera principal point.

A rigid transformation $\mathcal{T}_{\mathcal{V}\to\mathcal{C}}$ in 3-space further maps points in vehicle reference system and camera system. This is equivalent to a rotation and a translation, and

thus has 6 degree of freedom – otherwise known as *extrinsic parameters*.

$$\mathcal{T}_{\mathcal{V}\to\mathcal{C}} = \begin{pmatrix} \mathbf{R}_{\mathcal{VC}} & | & \mathbf{t}_{\mathcal{VC}} \end{pmatrix} \tag{2.2}$$

where $\mathbf{R}_{\mathcal{VC}}$ is a 3×3 rotation matrix, and $\mathbf{t}_{\mathcal{VC}}$ is a translation vector in 3-space. The full transformation can be formulated as,

$$\tilde{\mathbf{x}}_{\mathcal{C}} = \mathbf{K}_{\mathcal{C}} \mathcal{T}_{\mathcal{V}\to\mathcal{C}} \tilde{\mathbf{x}}_{\mathcal{V}} \tag{2.3}$$

where $\tilde{\mathbf{x}}_{\mathcal{V}} = (x\ y\ z\ 1)^T$ is a 3D point in vehicle homogeneous system $\tilde{\mathcal{V}}$, and $\tilde{\mathbf{x}}_{\mathcal{C}} = (\tilde{u}\ \tilde{v}\ \tilde{w})^T$ denotes the homogeneous coordinates of projected pixel. The inhomogeneous (Euclidean) coordinates for the projected pixel is computed as $\mathbf{x}_{\mathcal{C}} = [\tilde{u}/\tilde{w}\ \tilde{v}/\tilde{w}]$.

LiDAR points are mapped to vehicle system $\mathcal{V}$ in a similar fashion. Range measurements are first converted from Spherical to Cartesian system. Then they are mapped using a 3×4 rigid transformation $\mathcal{T}_{\mathcal{V}\to\mathcal{L}} = \begin{pmatrix} \mathbf{R}_{\mathcal{VL}} & | & \mathbf{t}_{\mathcal{VL}} \end{pmatrix}$ in 3-space,

$$\mathbf{x}_{\mathcal{L}} = \mathcal{T}_{\mathcal{V}\to\mathcal{C}} \tilde{\mathbf{x}}_{\mathcal{V}} \tag{2.4}$$

with $\tilde{\mathbf{x}}_{\mathcal{V}}$ being a homogeneous point in vehicle system $\tilde{\mathcal{V}}$, and $\mathbf{x}_{\mathcal{L}}$ denoting the associated inhomogeneous point in LiDAR reference system $\mathcal{L}$. Vehicle-to-laser transformation can then be estimated as,

$$\mathcal{T}_{\mathcal{V}\to\mathcal{L}} = \mathcal{T}_{\mathcal{V}\to\mathcal{C}} \times \mathcal{T}_{\mathcal{C}\to\mathcal{L}} \tag{2.5}$$

These parameters are calculated during an offline [37, 48, 119] or online (scene induced) [93] calibration process. However, even with accurate calibration, these transformations may map measurements to irrelevant positions. Since LiDAR

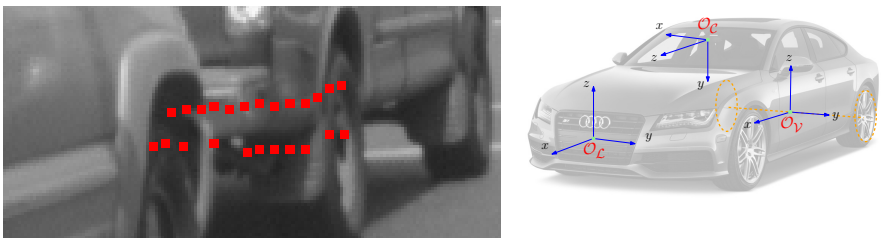


Figure 2.3: An example of parallax, caused by the distance between sensors. In this case, LiDAR is mounted on front bumper, and Camera is mounted behind the windshield.

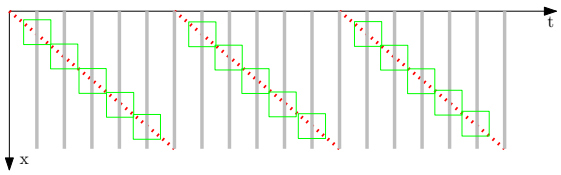


Figure 2.4: A depiction of measurement-level (discrete) association to minimize drift. LiDAR points (in red) in one scan have different timestamps, and they can be associated to different images (in gray) to decrease sample time different and motion drift.

and camera sensors are mounted on different locations on vehicle body, they have different perspectives of the scene. This is referred to as parallax, and one such scenario is shown in Fig. 2.3. The laserscanner (mounted on front bumper in this case) receives reflections from a target vehicle that is partially occluded in the image. This indicated that the continuous mapping between different sensor spaces is not perfectly linear and actually depends on the structure of the scene. Joint analysis of all obstacles in the scene and depth-aware ordering can potentially alleviate this issue. This problem is studied in more depth in the next chapters.

2.3.2 Discrete Multi-modal Measurement Association

One fundamental requirement for processing two sensors is for them to measure the same underlying latent random variable. This allows us to assume the measurements coming from different sensors contribute to the same likelihood

function. One possible way to guarantee this is to ensure the acquisition time difference Δt between measurements is close to zero. If the underlying object has velocity $\mathbf{v}$, then it has a drift $\mathbf{v}\Delta t$ between sensors. This spatial drift leads to erroneous association of measurements cross sensor spaces, and violates the assumption of common latent variable. As will be discussed in Chap. ch:estimation, one can model this drift using initial estimated velocity, refine it by shifting measurements by $-v\Delta t$, updating associations, and estimating velocity in an iterative manner. Although this serves as a robust remedy for the drift issue, it requires additional steps and computations and works only if the drift is small. If the drift is too large, chances are the initial estimated velocity has a large bias, and the correction process is not convergent.

It is possible to reduce the effect of the drift through association of discrete samples. This approach is simply depicted in Fig. 2.4. LiDAR point clouds (shown in red) are collected as the internal mirror or transmitting device is rotating, and in typical devices this takes around 100ms. As a result, readings within one scan have different timestamps on the horizontal axis. One way to minimize Δt between associated points and pixels is to associate data at measurement-level rather than cloud-level. The green boxes show association according to measurement-level approach.

2.4 Distance-aware Processing Resolution

The two modalities we are working with in this problem have completely different natures, so unless a reliable and common measurement model is used they cannot be fused. The geometric transformations applied to these measurements also change their distribution in the target space. Therefore, sensor fusion should combine estimates considering their transformed distribution. The influence of

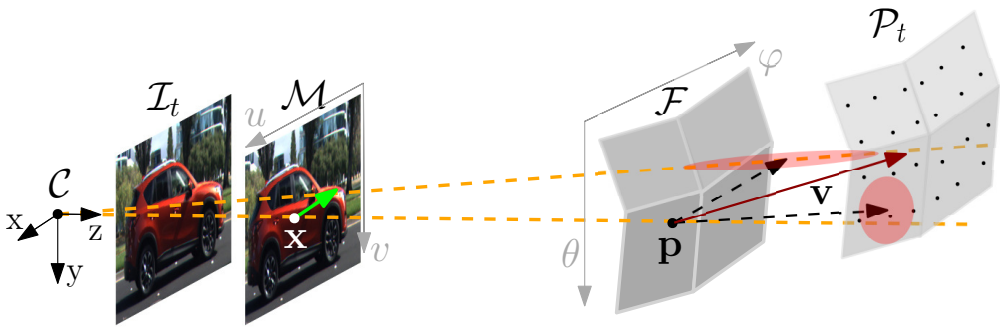


Figure 2.5: A representation of how intermediate surface $\mathcal{F}$ is employed to both convert image-plane displacements to measurements on velocity and facilitating point-set registration. Red ellipsoids represent the uncertainty in measurements.

such transformations is studied in this part.

We start by calculating the pixel-wise distance between projections of adjacent LiDAR points onto the image plane. We know LiDAR's scanning pattern has a fixed horizontal and vertical angular distance, denoted respectively by $\Delta\varphi$ and $\Delta\theta$. We consider two horizontally adjacent LiDAR points, and assume they have relatively the same range ρ. Then the horizontal component of their physical displacement can be computed as,

$$d_X = \rho\Delta\varphi \tag{2.6}$$

Assuming camera images are undistorted and calibration parameters are available, we find the pixel-wise distance on image plane as,

$$d_x = f_x\frac{d_X}{Z} = f_x\frac{\rho\Delta\varphi}{\rho cos(\varphi)} \approx f_x\Delta\varphi \tag{2.7}$$

which is approximated for objects straight ahead with $cos(\varphi) \approx 1$. Similar calculations show vertical distance between points can be approximated as $f_y\Delta\theta$. One key observation is that the density of projected LiDAR points onto image plane

Figure 2.6: When LiDAR measurements are projected onto the image plane, they are roughly distributed evenly independent of their range.

does not actually depend on ρ. Although closer regions will have more LiDAR samples due to diverging rays, but projections onto image plane are almost evenly distributed. For our experiments with KITTI dataset, pixel-wise horizontal and vertical distances are respectively 2.09 and 5.55 pixels. An example of LiDAR points from KITTI dataset projected onto image plane is shown in Figure 2.6.

We continue this discussion by a high-level overview of fusing LiDAR-based and image-based measurements on velocity. Visual features on motion are typically derived from *brightness constancy assumption* – which states that under the right mapping pixels in two consecutive video frames should be similar. It is possible to represent a visual motion measurement from pixel i as,

$$z_i = \mathbf{h_i}^T \mathbf{x} + n_i, \qquad i \in \{1, 2, \cdots, N\} \tag{2.8}$$

where z_i is a scalar measurement, $\mathbf{h_i}$ is a 3-vector measurement model, $\mathbf{x}$ is the two-dimensional (pixel-wise) displacement vector on image plane, and n_i is a zero-mean additive noise. Also N denotes the total number of pixels for a single object. For the sake of simplicity, we assume n_i is distributed as a zero-mean Gaussian random variable with standard deviation σ_n. As explained in Chapter 3, we can relate $\mathbf{x}$ to the actual three-dimensional velocity $\mathbf{v}$ in the physical environment through a per-pixel matrix $\mathbf{B_i} = f/Z_i\mathbf{B}$. Note $f = f_x \approx f_y$ is the focal length and

Z_i is pixel depth. Thus we can represent the visual measurement in terms of $\mathbf{v}$ as,

$$z_i = \frac{f}{Z_i}\mathbf{h_i}^T\mathbf{B}\mathbf{v} + n_i, \qquad i \in \{1, 2, \cdots, N\} \tag{2.9}$$

As a result, z_i has a Gaussian distribution with mean $f/Z_i\mathbf{h_i}^T\mathbf{B}\mathbf{v}$ and standard deviation σ_n. In addition to visual measurements, we have velocity observations from LiDAR point clouds. Using the Generalized ICP [94] framework, we can describe them as,

$$\mathbf{y_i} = \mathbf{F_i}\mathbf{v} + \mathbf{n_i}, \qquad i \in \{1, 2, \cdots, M\} \tag{2.10}$$

with M denoting total number of LiDAR points on the object. Note that $\mathbf{y_i}$ and $\mathbf{n_i}$ are both 3-vectors, and noise has a Covariance matrix of $\sigma_m^2\mathbf{I}$. Thus a total of $N + M$ measurements are available on velocity, forming an overcomplete system of linear equations. A typical approach to solve the inverse problem is to form the negative log likelihood function for $\mathbf{v}$,

$$\begin{aligned} -logL(\mathbf{v}) &= -log\Big(\prod_{i=1}^{N} p(z_i; f/Z_i\mathbf{h_i}^T\mathbf{B}\mathbf{v}, \sigma_n)\prod_{i=1}^{M} p(\mathbf{y_i}; \mathbf{F_i}\mathbf{v}, \sigma_m\mathbf{I})\Big) \\ &\propto \sum_{i=1}^{N}\frac{1}{2\sigma_n^2}\|z_i - \frac{f}{Z_i}\mathbf{h_i}^T\mathbf{B}\mathbf{v}\|^2 + \sum_{i=1}^{M}\frac{1}{2\sigma_m^2}\|\mathbf{y}_i - \mathbf{F}_i\mathbf{v}\|^2 \\ &= \sum_{i=1}^{N}\frac{1}{2\sigma_n^2}(\frac{f^2}{Z_i^2}\mathbf{v}^T\mathbf{B}^T\mathbf{h_i}\mathbf{h_i}^T\mathbf{B}\mathbf{v} + \cdots) + \sum_{i=1}^{N}\frac{1}{2\sigma_m^2}(\mathbf{v}^T\mathbf{F_i}^T\mathbf{F_i}\mathbf{v} + \cdots) \end{aligned} \tag{2.11}$$

where first and zeroth order terms are not shown. The quadratic term in the likelihood function describes the underlying uncertainty in the Maximum Likelihood estimate of the latent variable $\mathbf{v}$. This method is extensively studied in Chap.

3, but here we focus on the relationship between some of the hyper-parameters and the resulting uncertainty in velocity. Let's assume we want to tune hyper-parameters so that objects, regardless of their distance have the same velocity uncertainty ϵ. For example, objects that are closer to sensors will have more measurements, and this results in more accurate velocity estimation. But this comes with the cost of processing more samples, while it might not be necessary if estimation meets the accuracy threshold ϵ. We further simplify the formulation by assuming all pixels belonging to the object have the same depth as Z_0 and all observation matrices are equal to $\mathbf{h}$ or $\mathbf{F}$. We also note that the number of object pixels N and object points M is proportional to inverse squared of distance; it is, therefore, safe to denote them as $N = \tilde{N}/Z_0^2$ and $M = \tilde{M}/Z_0^2$. To formalize this, we expand the precision matrix of $\mathbf{v}$ as,

$$\begin{aligned}\mathbf{P_v} &= \sum_{i=1}^{N} \frac{f^2}{Z_0^2 \sigma_n^2} \mathbf{B}^T \mathbf{h}\mathbf{h}^T \mathbf{B} + \sum_{i=1}^{M} \frac{1}{\sigma_m^2} \mathbf{F}^T \mathbf{F} \\ &= \frac{N f^2}{Z_0^2 \sigma_n^2} \mathbf{\Gamma} + \frac{M}{\sigma_m^2} \mathbf{\Lambda} = \frac{\tilde{N} f^2}{Z_0^4 \sigma_n^2} \mathbf{\Gamma} + \frac{\tilde{M}}{\sigma_m^2 Z_0^2} \mathbf{\Lambda}\end{aligned} \tag{2.12}$$

where $\mathbf{\Gamma} = \mathbf{B}^T \mathbf{h}\mathbf{h}^T \mathbf{B}$ and $\mathbf{\Lambda} = \mathbf{F}^T \mathbf{F}$ depend on the observation model. In order to determine the perfect resolution at which pixels and points need to be processed, we consider a discrete pyramid. Consider a Gaussian pyramid with downsampling factor of 2 and $l = 0$ denoting the full-resolution grid. With increasing l the grid is smoothed (to prevent aliasing) and downsampled. The task of finding the optimal processing resolution is then to find the optimal l. One should note that by going to the next pyramid level, number of points and pixels belonging to each object is divided by four; so, $\tilde{N} = N_0/4^l$ and $\tilde{M} = M_0/4^l$. Camera focal length f also depends on pyramid level, and can be rewritten as $f = f_0/2^l$. Substituting the

new variables in Eq. 2.12,

$$\mathbf{P_v} \propto \frac{N_0 f_0^2}{(Z_0^2 4^l)^2 \sigma_n^2}\mathbf{\Gamma} + \frac{M_0}{(Z_0^2 4^l)\sigma_m^2}\mathbf{\Lambda} \tag{2.13}$$

In the expression above, all variables are constant except for $Z_0 2^l$. In order for the precision in Eq. 2.13 result in approximately the same uncertainty ϵ for all objects regardless of their distance Z_0, pyramid processing level l should be chosen such that the term $Z_0 2^l$ is roughly constant,

$$Z_0 2^l = k \Rightarrow l = lgk - lgZ_0 \tag{2.14}$$

where k is a constant. This expression suggests moving to the next finer pyramid level when object distance is doubled. The idea of distance-aware processing level is depicted for a simple scenario in Fig. 2.7. Target is processed at the finest pyramid level when it is far, and at coarser levels when it is closer to the camera. There is another motivation behind distance-aware processing level, and it is rooted in the linearization step of brightness constancy assumption. Due to this Taylor expansion approximation, the displacement vector on image plane needs to be small relative to pixel size. In order to ensure this assumption holds, many optical flow computation methods rely on coarse-to-fine pyramid-based processing since at coarser resolutions the apparent displacement vector is smaller by a factor of 2^l.

We follow the same strategy in our method, while noting distance-based ways to quantify displacement vectors. Let's assume we have an upper-bound $\|\mathbf{v}_{max}\|$ on the magnitude of relative velocity vector that objects in the scene could attain. With ψ denoting the angle between object direction and line of sight, we can express the magnitude of the corresponding apparent image-plane displacement

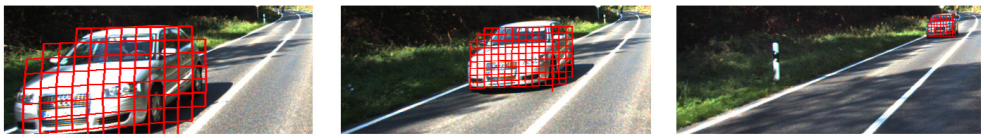


Figure 2.7: Processing of objects at different pyramid resolutions depending on their depth. Closer objects (left) are processed at coarser pyramid levels, leading to roughly equal number of cells per object.

as,

$$\|\mathbf{x}_{max}\| = \frac{\Delta t f}{Z_0}\|\mathbf{v}_{max}\| cos\psi \leq \frac{\Delta t f_0}{Z_0 2^l}\|\mathbf{v}_{max}\| \tag{2.15}$$

which has to be smaller than a threshold $\delta \approx 1px$ in order for the Taylor approximation to hold. In simple terms, the upper bound in the expression above defines our spatial search space (on the image plane) to match each pixel in the previous frame. There are two fundamental ways to guarantee that search space is small enough: decreasing Δt, or increasing processing level l. It's important to note the term $Z_0 2^l$ in denominator is equal to k in Eq. 2.14. Therefore, closer objects should naturally be processed at coarser levels to reduce the search space. In some sensing devices it is possible to compromise the scanning rate and resolution, or sampling rate may be higher than needed. In such cases we can choose the rate at which samples are processed in each step. Ideally we want to choose Δt small enough so that the search space is covered. But setting Δt too small could result in low velocity precision, as both $\mathbf{\Gamma}$ and $\mathbf{\Lambda}$ in Eq. 2.13 are inversely proportional to Δ_t^2. Therefore the frame rate needs to be chosen to balance both effects.

2.5 Proposed Fusion Architecture

The proposed environment perception module, as shown in Figure 2.8, can be broken down into two primary sub-modules: (i) measurement grouping, and (ii)

velocity estimation and surface reconstruction. In this section, we briefly describe the role of each sub-module and their interactions.

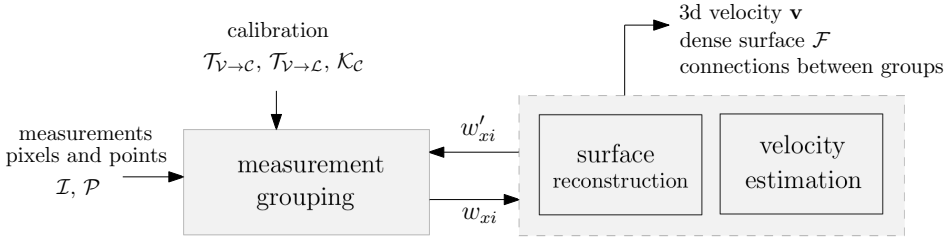


Figure 2.8: High-level block diagram of our environment perception module

Measurement Grouping

A critical step in jointly processing images and point clous is to jointly cluster them into higher-level groups. Each group is linked to a latent variable and constitutes a constant-motion unit. Fig. 2.9 shows a spectrum of constant-motion units employed in the literature, with green dots denoting the underlying latent variables. Clustering measurements into more abstract levels (right side of the spectrum) is necessary unless we intend to estimate an independent velocity vector for each point and pixel [89] (left side of the spectrum). Examples of constant-motion units are raw pixels [31, 89, 88], occupancy grids [74, 112], superpixels [116, 42, 108, 69], and thin vertical columns [26] otherwise known as Stixels. A low degree of abstraction results in unnecessary computations, since it neglects the redundancy of most neighboring points/pixels moving with the same velocity. Once we have a rough grouping of measurements into coherently moving clusters, we can impose motion constraints on all pixels and all points that belong to the group. At the same time, object-level abstraction is very challenging and in some cases unreliable. First of all, it is computationally expensive to perform flawless object-level segmentation. Also, in most cases trained models are necessary for

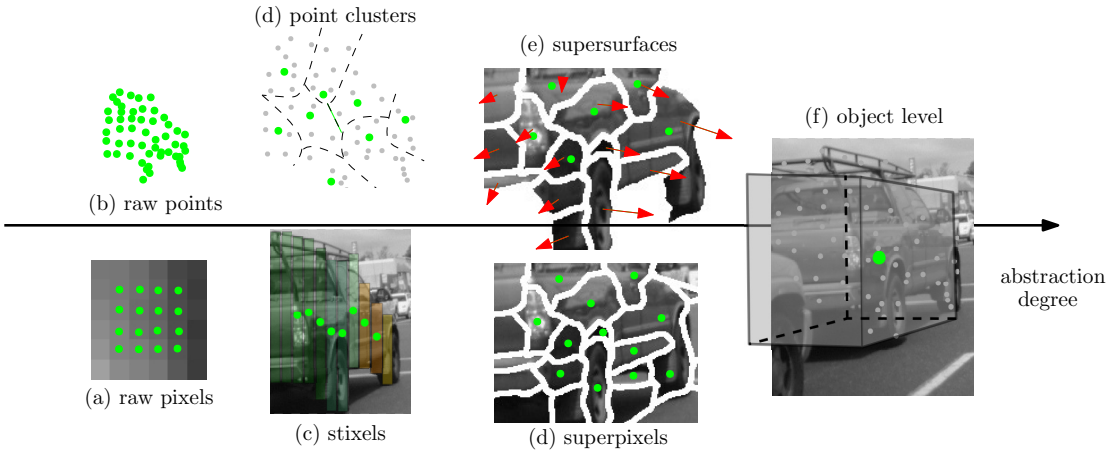


Figure 2.9: Different abstraction degrees

object-level abstraction and we do not want to rely on object classification. In fact, pseudo-object-level abstraction can be easily achieved through modeling the connections and interactions between nearby segments [108].

Another approach is to model the environment as a set of superpixels with underlying slanted plane surface model [116] and use them as constant-motion units. Optical flow and depth observations falling inside each superpixel are processed jointly, resulting in coherent motion estimates for each segment [108] or each object [69]. Their work also takes into account the interactions between neighboring superpixels. We follow a similar path and use Supersurfaces as our constant-motion units. As explained in detail in Chapter 4, they are depth-adapted three-dimensional surfaces along with their projected two-dimensional mask on the image domain. It is worth mentioning that the association of each measurement (pixel or point) the clusters is not binary. Assignments are modeled as a real number between zero and one. Depth-adapted means, unlike conventional superpixel methods, their expected size varies with depth, smaller segments for farther objects and larger segments for closer ones. This is aligned with the intuitions explained in Section X regarding estimation-theoretic and geometrical constraints.

Surface Reconstruction

Fitting an intermediate surface to each segment (constant-motion unit) helps assign dense depth values to image pixels and mitigate the lower relative resolution of LiDAR. Per pixel depth is necessary in order to convert pixel-wise displacement vectors into metric constraints on physical velocity. For a camera with known intrinsics, we can infer the component of metric velocity that is in image plane if we know pixel depth [86]. This is studied in depth in Chap. 3. In simple terms, the reconstructed surface is the key to fuse the measurement space of LiDAR and Camera.

Due to low resolution sampling of most LiDARs, direct matching of LiDAR points across multiple scans leads to biased (and confident) velocity estimates. This is especially critical about long-distance targets with complex and rough surfaces, as the Nyquist sampling criteria is more likely to be violated. A reconstructed surface can also be employed for registering scans of a single object, and to alleviate inaccuracies of point-to-point registration. This is similar to Generalized ICP [94] or point-to-plane matching. Also, LiDAR measurements are erroneous, noisy, have outliers, suffer from occlusions and contain missing data. A robust surface can also help alleviate outliers, reduce noise power, and cope with missing data.

The choice of surface representation greatly depends on the choice of sensor and the shape prior knowledge. It is also influenced by constant-motion units and our criterion for grouping measurements. Choices range from very restricted models such cuboids [78, 55], to less restricted models like linear combination of basis functions [59], mixture of slanted planes, boundary points [105, 110], stixels [26], or zero-level-set of a 3-D embedding function [81, 80]. While models as simple as cuboids can't properly model complex object like a cyclist, model-

free representations are too expensive to work with, extremely non-convex, and without continuous first and second order derivatives. This makes them unpopular to be applied in energy-based minimization methods.

In Chap. 3 we model surfaces as a linear combination of B-spline basis functions parametrized by a set of coefficients. This representation easily accommodates surface priors like piece-wise planar regularization. This is achieved by penalizing (pre-computed) first or second order derivatives of the basis functions. Later in Chap. 5 and for CUDA implementation purposes, we replace surface reconstruction with a data-adaptive depth extrapolation technique.

Velocity Estimation and Tracking

This module jointly processes the point cluster $\mathcal{P} = \{(\mathbf{p}_i, t_i)\}$ and pixel cluster $\mathcal{I} = \{(\mathbf{x}_i, t_i)\}$ as measurements of each group (cluster). The result is the instantaneous velocity vector $\mathbf{v}$ of the group, along with velocity Covariance matrix $\mathbf{C}_\mathbf{v}$. For each measurement in $\mathcal{P} \cup \mathcal{I}$ it also provides the likelihood that measurement actually belongs to the underlying group. In other words, the estimation unit is robust against outliers (both pixels and points), and is able to refine object clusters by updating the membership weights.

Optical flow, i.e. inferring pixel motions (u, v) from a pair of images $\{\mathcal{I}_S, \mathcal{I}_T\}$, is based on brightness constancy assumption between images. This assumption is violated under several conditions, e.g. occlusion, reflections, varying lighting, multiple moving objects, and appearance variations like car indicators.

Chapter 3

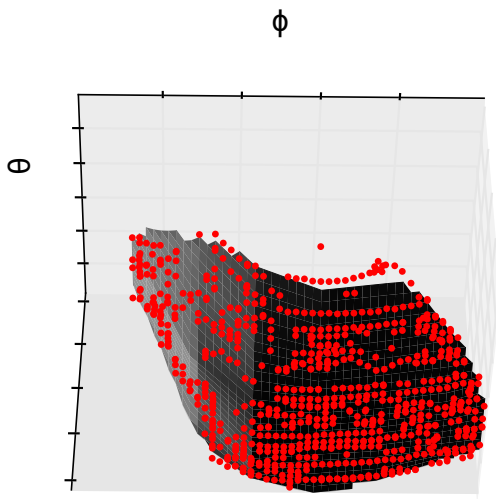
Velocity Estimation

In this chapter, we explain our object-class-agnostic tracking and reconstruction algorithm for rigid obstacles with arbitrary shape and appearance. This is opposed to tracking-by-detection [35] in which objects of one class, e.g. cars, are detected and tracked. It is also different from loosely-coupled (high-level) fusion, in which object candidates or tracks are generated separately for each sensor and merged in the last step [98, 73]. Measurement grouping is assumed to be solved throughout this chapter.

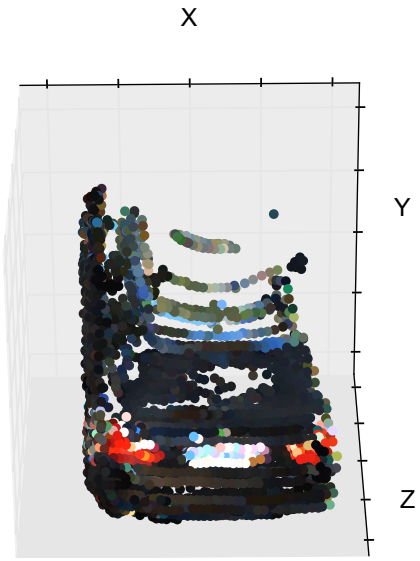
In order to estimate object velocities, we directly use raw visual and range sensor measurements in a differential and feature-less energy minimization framework. This is achieved by directly minimizing a multi-modal cost function. An intermediate object representation $\mathcal{F}$ is also proposed, shown in Fig. 3.1c, that relates sparse 3D LiDAR data in spherical system to dense Camera data in Cartesian 2D system. As explained in Chap. 2, there are fundamental challenges in fusing raw LiDAR and camera measurements. First, they are different in nature, e.g. image data is dense, while LiDAR point clouds are sparse angle measurements that contain range and intensity. Second, they may have different sampling rates and time-stamps. Third, since these sensors are mounted at different positions

(a) Raw measurements

(b) Estimated 3D $\mathbf{v}$



(c) estimated surface, $\mathcal{F}$



(d) accumulated points, $\mathcal{P}^{\star}$

Figure 3.1: Our tracking and reconstruction (a) takes sequences of LiDAR point clouds and camera images, and (b) finds 3D velocity $\mathbf{v}$ for each object. It (c) estimates an intermediate surface $\mathcal{F}$ to fuse sensors, and (d) could potentially accumulates points as a 3D model.

they are susceptible to parallax. Yet another challenge is to efficiently represent objects in both sensor spaces. To the best of our knowledge, this work is the first to address these challenges and to lay a dense, multi-modal and measurement-level LiDAR and camera fusion method for object tracking and reconstruction.

3.1 Related Work

In this section we study significant prior work in the areas of point cloud matching, scene flow estimation, and tightly-coupled sensor fusion for motion inference.

Traditional Point Cloud Registration Techniques

Iterative Closest Point (ICP) [10] is a point-cloud registration method that minimizes the distance between two point sets by finding a rigid transformation $\mathcal{T} = (\mathbf{R}|\mathbf{t})$. Let $A = \{a_i\}$ and $B = \{b_i\}$ be source and template point sets. At each iteration, Standard (point-to-point) ICP refined $\mathcal{T}$ as,

$$\mathcal{T} \leftarrow \arg\min_{\mathcal{T}} \Big\{ \sum_i w_i \Big\| \mathcal{T} b_i - \underbrace{\arg\min_{a_j} (\|\mathcal{T} b_i - a_j\|^2)}_{m_i} \Big\|^2 \Big\} \tag{3.1}$$

where the inner minimization finds the closest point in A for each transformed point $\mathcal{T} b_i$ in B, and the outer minimization refines $\mathcal{T}$. Therefore, $\mathbf{R}$ and $\mathbf{t}$ are refined at each iterations as the MLE solution to (3.1). When implemented with kd-trees ICP-based methods have complexity $\mathcal{O}(n \log n)$. Point-to-plane ICP [20], considers minimizing the distance between points and local planes; in particular, it minimizes,

$$\mathcal{T} \leftarrow \arg\min_{\mathcal{T}} \Big\{ \sum_i w_i \Big\| \eta_i . \Big(\mathcal{T} b_i - m_i \Big) \Big\|^2 \Big\} \tag{3.2}$$

where η_i is surface normal vector at m_i. 3.2 assumes η_is can be estimated, which with our sparse measurements such estimates - even with accumulated clouds - are not nearly accurate. There are other ICP variants, e.g. Generalized ICP [94], which considers a plane-to-plane metric, PL-ICP [17] with a point-to-line metric, LM-ICP [30] which employs Levenberg-Marquardt for nonlinear optimization, Sparse-ICP [68] which considers ℓ_p norm with $p \in [0, 1)$. But most of these methods are geared toward registering point clouds that are dense or contain many points. When, as in our setup, point sets are too sparse with very few number of points most of these methods will suffer from rank deficiency.

Kernel Correlation (KC) [102] is a kernel based point-set registration that, unlike ICP that finds point-wise correspondences, considers all point pairs in the objective function. Tsin et al, [102] propose to minimize a distance kernel $KC(a_i, b_j)$ summed over all pairs of points (a_i, b_j). With a Gaussian kernel in place, at each iteration, KC minimizes,

$$\mathcal{T} \leftarrow \arg\min_{\mathcal{T}} \sum_{a_i \in A} \sum_{b_j \in B} (\pi\sigma^2)^{-3/2} exp\Big\{ - \|a_i - b_j\|^2/\sigma^2 \Big\} \tag{3.3}$$

where σ defines the extension or kernel size which is incrementally decreased. This will help to recover large motions first, without getting stuck in local minima. Then finer adjustments are made with lower values of σ. While this inherently has complexity $\mathcal{O}(N^2)$ in the number of points, one can precompute coarse-to-fine 3D grids and then maximize correlation without the need to recompute distances. There are KC variants that couple it with Gaussian Mixture Models (GMM), e.g. GMM-KC [54] or Coherent Point Drift (CPD) [72]. It is possible to employ the rank-deficiency of $N \times N$ matrix to drop the complexity to $\mathcal{O}(N \log N)$ [72]. In addition, CPD uses the Fast Gauss Transform to approximate sums of Gaussian kernels in $\mathcal{O}(M + N)$, where M is the number of grid points. While KC-based

methods are very successful at aligning dense and/or large point clouds, they have similar poor performance as ICP when it comes to sparse clouds with very few points.

Piecewise Planar Scene Flow

Scene flow, first introduced by Vedula et al, [106], is a 3D motion vector defined at each pixel, that combines image optical flow with depth change. These features are computed in a sparse [60, 31] or dense [111, 87] manner and primarily using stereoscopic vision setups. Rabe and Franke first presented 6DVision [31], which computes sparse scene flow at distinctive pixels. In their Dense6D [89] system, they track each pixel in 3D using a Kalman filter and based on precomputed dense optical flow and stereo. Yamaguchi et al, [116] use a piece-wise planar model and break the scene into superpixels. They form a Markov Random Field (MRF) with a static scene assumption, then optimize segments, ego-motion, occlusion, and plane parameters. Vogel et al, [108] find plane parameters for each segment too, but also consider a pairwise term between nearby planes. In order to deal with dynamic scenes, they compute a rigid motion for each segment. Menze et al, [69] group segments into objects, and find motion parameters for each object rather than all segments. Quiroga et al, [87, 86] take two consecutive intensity and depth maps from an RGBD camera, and track pixels in both intensity and depth using Lucas-Kanade. Sun et al, [99] decompose depth map into layers, and minimize a Gaussian MRF to estimate dense scene flow. Hadfield et al, [44] use particle filters and resample from the joint depth and appearance posterior over time. Despite their promising results, most of stereo or RGBD-based scene flow computation methods take hundreds of seconds to process an image pair.

Odometry and Mapping using LiDAR and Camera

In [118] Zhang et al, propose V-LOAM which combines visual and LiDAR odometry. They employ visual and geometric feature matching to estimate ego-motion. The high-frequency visual odometry handles rapid motion, and LiDAR odometry warrants low-drift and robustness in undesirable lighting conditions. Although such feature-based methods have been employed and have produced promising results in Simultaneous Localization and Mapping (SLAM), they are not suited for multi-object tracking. Feature extraction techniques may fail to provide or match enough number of visual/geometric features for some objects, especially if the object is small, far, texture-less, or occluded.

Non-parametric object tracking

Several methods align tracked object point clouds with new segmented scans, then append the segment to and enrich the 3D model [114, 52, 46]. Wyffels and Campbell [114] lay a probabilistic method to estimate motion and shape for a single extended object using simulated LiDAR data, but do not experiment with real-world datasets and do not add camera. Held et al, [46] track 2D velocity vectors in real-time and accumulate points to form a dense model for each track. A latent surface variable is implicitly included in their velocity posterior, modeled as a collection of points sampled from the visible surface. They make use of a grid-based sampling method, annealed dynamic histograms, to maximize the velocity posterior. They start by sampling from the state space at a coarse resolution, using an approximation to the posterior distribution over velocities. As the resolution increases, they anneal this distribution so that it approaches the true posterior. Their method is capable of tracking in colored point clouds where RGB values are added to sparse points, but does not use dense image data directly and as a

separate modality.

Velocity Estimation using LiDAR and Camera

Ilg et al, [52] employ LiDAR and Camera, mounted on a stationary platform, to estimate pose and reconstruct the 3D model of a moving object. They directly compute dense optical flow $\mathbf{f}_i$ on images, using Large Displacement Optical Flow [14] – which is computationally expensive. They continue by projecting range finder points $\mathbf{p}_i$ onto images $\mathbf{x}_i$, and computing their position on the next image as $\mathbf{x}'_i = \mathbf{x}_i + \mathbf{f}_i$. Then they define a point-line constraint, restricting the 3D position of $\mathbf{p}'_i$ to the projection ray of $\mathbf{x}'_i$. This ray is represented in Plücker form as $\mathbf{L}'_i = (\mathbf{m}'_i, \mathbf{n}'_i)$, with moment $\mathbf{m}'_i$ and unit vector $\mathbf{n}'_i$). The Plücker line representation then yields the distance of an arbitrary point $\mathbf{x}$ to $\mathbf{L}'_i$ as,

$$d(\mathbf{L}'_i, \mathbf{x}) = \|\mathbf{x} \times \mathbf{n}'_i - \mathbf{m}'_i\|. \tag{3.4}$$

Their final objective functions finds a transformation $\mathcal{T}$ that minimizes,

$$\hat{\mathcal{T}} = \arg\min_{\mathcal{T}} \sum_i \|\pi\Big(\mathcal{T}\mathbf{x}_i\Big) \times \mathbf{n}'_i - \mathbf{m}'_i\|. \tag{3.5}$$

where $\pi(.)$ denotes the projection from homogeneous to Euclidean coordinates. They restrict their transformation to a twist $\mathcal{T} = exp(\xi)$. Then in a Iterative Reweighted Least Squares (IRLS) fashion and at each iteration, they linearize the twist as $exp(\xi) \approx \mathbf{I} + \xi$. In order to treat outliers, they also assign truncated Huber weights to points at every iteration. Finally they perform point-to-plane ICP, by estimating normal vectors η_i based on accumulated model. This is feasible because of large FoV of the nodding range finder.

While the method of Ilg et al, [52] robustly accumulates points, based on the

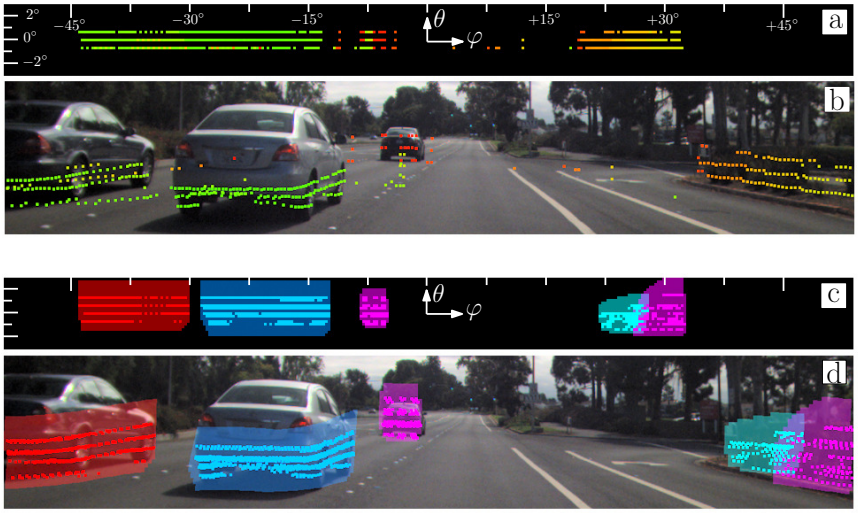


Figure 3.2: (a) raw LiDAR measurements on a $\theta\varphi$ grid in spherical coordinates with color encoding distance, (b) projection of points onto image, (c) estimated object domains $\mathcal{W}$ and accumulated points $\mathcal{P}^\star$, (d) projection of object domains onto image, i.e. $\mathcal{X}$.

quantitative results in the next section it fails to estimate motion with a fixed laser scanner – as opposed to their original tilting range finder. Our proposed approach has a similar nature to the one they present. But it handles multiple moving objects, does not require a fixed platform, a nodding range finder, or an expensive pre-computation of dense optical flow. It also optimizes all variables in a single-stage joint optimization framework, and does not need ICP-based post-processing.

3.2 Problem Formulation

The two primary sensors, camera and LiDAR, provide intensity (color) and depth measurements of the environment at different times. Let $\mathcal{P} = \{\mathbf{p}_1, \cdots, \mathbf{p}_N\}$ denote a cloud of points $\mathbf{p}_i = (\theta_i\, \varphi_i\, \rho_i)^T$ measured by laser scanner at t_i in spherical coordinates. The measurements are taken at discrete θ and φ intervals and in

LiDAR reference system, as in Fig. 3.2. Also let $\mathcal{I}(\mathbf{x})$ represent an intensity (or color) image captured by camera, and $\mathbf{x} = (x\ y)$. We assume camera intrinsic and extrinsic parameters (with respect to LiDAR) are calibrated. Therefore, as shown in Fig. 3.2, laser points and image pixels can be geometrically related. Let us also assume that a set of object candidates are provided as initial measurement groups. Upon arrival of a new image or laser sample at any time, our goal is to update velocity $\mathbf{v} = (v_x\, v_y\, v_z)^T$, 3D accumulated point cloud $\mathcal{P}^\star$, and surface model $\mathcal{F}$ for all objects. The estimation must be cross-modality, and performed in a robust Bayesian framework. Similar to [46] we assume $\mathbf{v}$ is pure translational (rotation-free) in short time periods. In order to model sensor uncertainties, we assume laser measurements have Gaussian noise with covariance,

$$\mathbf{C_p} = diag(\sigma_\theta^2, \sigma_\varphi^2, \sigma_\rho^2) \tag{3.6}$$

which is known from sensor characteristics.We also model images as Gaussian random variables with pixel-wise noise variance σ_I^2 across each channel.

3.3 Proposed Motion Estimation and Depth Reconstruction

Fig. 3.3 represents the overall tracking and reconstruction algorithm for one object. The object model parameters, shown in the middle column, are updated with new LiDAR (right side) or camera (left side) samples. In this section we introduce object model and the estimation approach.

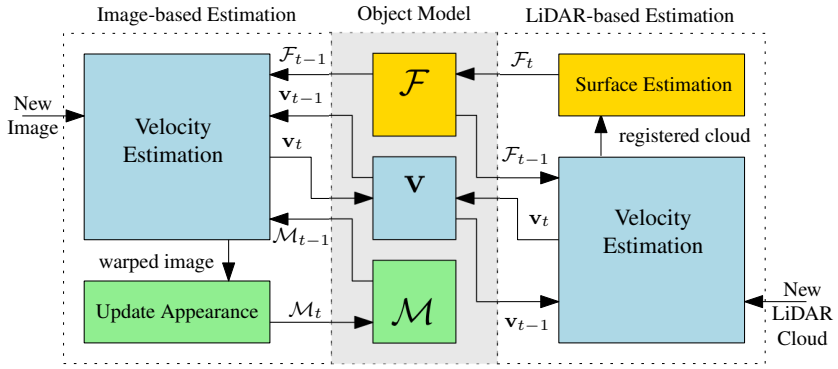


Figure 3.3: Block-diagram for the proposed velocity and surface estimation method. Objects are modeled with surface $\mathcal{F}$, appearance $\mathcal{M}$, and velocity $\mathbf{v}$.

3.3.1 Energy Function Derivation

In this section, we propose a new objective function that jointly optimizes object velocity and surface. The visible side of each object is modeled as an intermediate depth map $\mathcal{F}(\theta, \varphi) : \mathbb{R}^2 \rightarrow \mathbb{R}$ defined on an elevation-azimuth $(\theta\varphi)$ grid, and parametrized by $\mathbf{d}$. It is defined over a domain $\mathcal{W}$ (Fig. 3.2c) and is encouraged to be piecewise-planar. It can also be projected onto the image plane using calibration parameters, specifying a set of pixels $\mathcal{X}$ belonging to the object (Fig. 3.2d). This surface can be directly employed to match new laser measurements, it helps represent optical flow (on image plane) in terms of 3D velocity vector, and also makes possible explicit occlusion and parallax modeling. We also add an appearance model $\mathcal{M}$ for each object, which is simply an intensity (or color) template for the surface. As mentioned earlier, each object also has a unique 3D velocity vector, $\mathbf{v}$. Let $z_{1:t}$ denote the set of all sensor measurements, i.e. all images and laser scans ordered based on synchronized time-stamps. In the remainder of this section we focus on a single object. The posterior distribution

of velocity $\mathbf{v}_t$ can be written as,

$$\begin{aligned} p(\mathbf{v}_t|z_{1:t}) &= \int p(\mathbf{v}_{1:t}|z_{1:t}) d\mathbf{v}_{1:t-1} \\ &\propto \int p(z_t|\mathbf{v}_t) p(\mathbf{v}_t|\mathbf{v}_{t-1}) p(\mathbf{v}_{1:t-1}|z_{1:t-1}) d\mathbf{v}_{1:t-1} \\ &= p(z_t|\mathbf{v}_t) p_v(\mathbf{v}_t|z_{1:t-1}) \end{aligned} \tag{3.7}$$

The first term, referred to as likelihood term, captures the fidelity of z_t given state $\mathbf{v}_t$, and the second term is the prior (prediction) distribution for $\mathbf{v}_t$. Upon receiving a new image $\mathcal{I}_t$, the likelihood can be rewritten as,

$$p_C(\mathcal{I}_t|\mathbf{v}_t) = \int p_I(\mathcal{I}_t|\mathcal{M}_{t-1}, \mathbf{v}_t)\ p_M(\mathcal{M}_{t-1}) d\mathcal{M}_{t-1} \tag{3.8}$$

where p_I is image likelihood term given object appearance, and p_M is the appearance model. When a new laser scan $\mathcal{P}_t$ is received the likelihood will have a form,

$$p_L(\mathcal{P}_t|\mathbf{v}_t) = \int p_P(\mathcal{P}_t|\mathcal{F}_{t-1}, \mathbf{v}_t)\ p_F(\mathcal{F}_{t-1}) d\mathcal{F}_{t-1} \tag{3.9}$$

where p_P is the laser likelihood given surface model, and p_F is the surface distribution.

We represent the problem in its analogous energy form and address it as an energy minimization problem. Let E_F, E_C, E_L, and E_v represent energy forms for surface distribution, image likelihood, LiDAR likelihood, and velocity prior defined above. At each iteration, we first estimate the surface $\mathcal{F}$ based on accumulated object points $\mathcal{P}^\star$. As described in Sec. 3.3.2, surface $\mathcal{F}$ is represented

with a vector of coefficients $\mathbf{d}$. Surface energy E_F is then formulated as,

$$E_F(\mathbf{d}) = E_D(\mathbf{d}) + \lambda E_R(\mathbf{d}) \tag{3.10}$$

where E_D is surface data term based on $\mathcal{P}^\star$, E_R is surface regularization term, and λ is a constant. Surface coefficients $\mathbf{d}$, are updated by minimizing this energy function. Surface distribution is then modeled as a Gaussian with mean $\hat{\mathbf{d}}$. We can also estimate surface error variance, σ_F^2, based on surface energy term. When a new image is received, we have in energy form,

$$E(\mathbf{v}) = E_C(\mathbf{v}) + E_v(\mathbf{v}) \tag{3.11}$$

and update $\mathbf{v}$ by minimizing this energy. Then we update object appearance $\mathcal{M}$ as pixels specified by $\mathcal{X}$ from the new image. If the new sample is a laser point cloud $\mathcal{P}$ we have,

$$E(\mathbf{v}) = E_L(\mathbf{v}) + E_v(\mathbf{v}) \tag{3.12}$$

which is minimized in a similar fashion to update $\mathbf{v}$. Finally we add new object points from $\mathcal{P}$ to accumulated cloud $\mathcal{P}^\star$. Although the probability distributions defined above are not Gaussian, they can be approximated as locally Gaussian variables. This is analogous to employing an iterative energy minimization approach, e.g. Iterative Re-weighted Least Squares (IRLS) [49], and minimizing a quadratic function at each step. In what follows we explain each term in detail.

3.3.2 LiDAR Data Term $E_{\mathcal{P}}(\mathbf{v}, \mathbf{d})$

Object surface $\mathcal{F}$, defined on elevation-azimuth grid, is updated after each iteration based on object accumulated points $\mathcal{P}^\star$. Surface probability density

function is decomposed as,

$$p_F(\mathcal{F}_{t-1}|\mathcal{P}^\star) \propto p_D(\mathcal{P}^\star|\mathcal{F}_{t-1})\ p_R(\mathcal{F}_{t-1}) \tag{3.13}$$

with p_D being the data term, and p_R the regularization term enforcing piece-wise planar surfaces. Let us represent $\mathcal{F}$ as a linear combination of Q basis functions,

$$\mathcal{F}(\theta,\, \varphi) = \sum_{j=1}^{Q} d_j \phi_j(\theta,\, \varphi) \tag{3.14}$$

Let $\Omega_i = (\theta_i, \varphi_i)$ and r_i denote respectively the angular coordinates and range of adjusted laser point $\mathbf{p}_i$ in spherical coordinates. We assume p_D can be locally approximated as a Gaussian distribution at each optimization iteration. Then the log-likelihood of p_D at each iteration is defined as a quadratic function of $\mathbf{d} = (d_1, \cdots, d_Q)^T$ as,

$$E_D(\mathbf{d}) = \sum_{\mathbf{p_i} \in \mathcal{P}^\star} \frac{w_i}{\tau_i^2} \| \sum_{j=1}^{Q} d_j \phi_j(\Omega_i) - r_i) \|^2 \tag{3.15}$$

where w_i is Huber weight [49] computed from previous iteration error, and τ_i^2 is the range variance of $\mathbf{p}_i$. Note that $\mathcal{P}^\star$ consists of points accumulated from different times. For points in the last laser scan τ_i^2 is equal to the characteristic range variance σ_ρ^2. But older points, due to accumulation of velocity errors, have larger uncertainties and are given less weights. Huber weights also decrease over IRLS iterations for outliers.

3.3.3 Surface Smoothness Term $E_\mathcal{S}(\mathbf{d})$

This term encodes our prior knowledge of typical surfaces in an urban driving scene, e.g. vehicle bodies, ground, poles, buildings etc. As shown in [116, 108]

and other works, most surfaces in a driving scenario can be modelled piecewise planar. Inspired by the smoothness term in [59, 15], we minimize surface second-order derivatives, $\mathcal{HF}$, to encourage smooth planes. Nevertheless, we allow the surface to potentially bend at sparse patterns to prevents over-smoothed surfaces. In their depth reconstruction method, Calaki et al, [15] propose the smoothness term,

$$E_{\mathcal{S}}(\mathcal{F}) = \int_{\Omega} \|\mathcal{HF}(\theta, \varphi)\|_F \tag{3.16}$$

where $\|.\|_F$ is the Frobenius norm, $\mathcal{H}$ denotes Hessian, and Ω is the domain of angular grid θ, φ. The energy in (3.16) is the second-order derivatives of the height field summed over entire angular grid θ, φ. As in Klowsky et al [59], this energy can be discretized and represented in terms of $\mathbf{d}$ as,

$$E_{\mathcal{S}}(\mathbf{d}) = \mathbf{d}^T \mathbf{Q_s} \mathbf{d} \tag{3.17}$$

where,

$$\Big[\mathbf{Q_s}\Big]_{\alpha,\beta} = \sum_{\theta,\varphi \in \Omega} < \mathcal{H}\phi_\alpha(\theta, \varphi), \mathcal{H}\phi_\beta(\theta, \varphi) > \tag{3.18}$$

where $< ., . >$ denotes the sum of the element of the Hadamard multiplication of Hessians. We are, by building this matrix, representing the regularization as a quadratic term.

$$< \mathcal{H}\phi_\alpha, \mathcal{H}\phi_\beta >= \frac{\partial^2 \phi_\alpha}{\partial \theta^2} \frac{\partial^2 \phi_\beta}{\partial \theta^2} + 2 \frac{\partial^2 \phi_\alpha}{\partial \varphi \partial \theta} \frac{\partial^2 \phi_\beta}{\partial \varphi \partial \theta} + \frac{\partial^2 \phi_\alpha}{\partial \varphi^2} \frac{\partial^2 \phi_\beta}{\partial \varphi^2} \tag{3.19}$$

Note that all of the basis function along with their derivatives and $\mathbf{Q_s}$ can be precomputed and hard-coded in the program. Although minimizing this objective function gives semi-smooth surfaces, but it also leads to over-smoothed edges as the regularization term penalized the whole domain Ω equally. In addition, the

ℓ^2 norm of the error makes the data term sensitive to outliers. We continue by proposing a new objective function,

$$E_{\mathcal{S}}(\mathbf{d}) = \sum_{\theta,\varphi} \rho_2(\underbrace{\sqrt{\|\mathcal{H}\mathcal{F}(\theta,\varphi)\|_F}}_{e_{\theta\varphi}}) \tag{3.20}$$

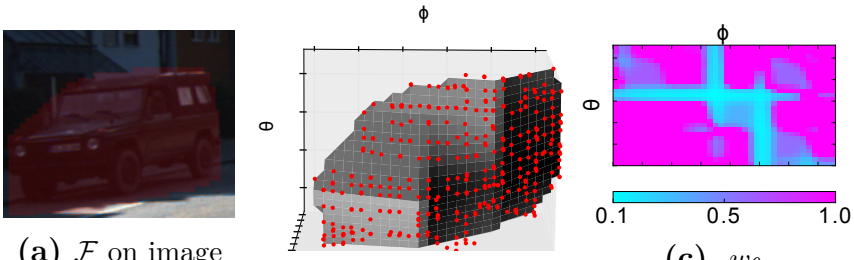
where $\rho_2(.)$ denotes Huber function with threshold τ, and $e_{\theta\varphi}$ is the residual for voxel (θ, φ). Most (θ, φ) voxels have large weights, resulting in stronger penalization of second-order derivatives. But these weights are small for a sparse set (pattern) of voxels, which leads to less regularization and possibly surface bending. The reweighted quadratic energy function at each IRLS iteration is,

$$E_{\mathcal{S}}(\mathbf{d}) = \sum_{\theta,\varphi\in\Omega} w_{\theta\varphi} \|\mathcal{H}\mathcal{F}(\theta,\, \varphi)\|_F = \mathbf{d}^T\mathbf{\Gamma}^T\mathbf{W}_{\theta\varphi}\mathbf{\Gamma}\mathbf{d} \tag{3.21}$$

where $\mathbf{W}_{\theta\varphi}$ is the diagonal matrix of regularization weights, and Γ is a tensor that performs Hadamard multiplication with second-order derivatives. $\mathbf{\Gamma}^T\mathbf{W}_{\theta\varphi}\mathbf{\Gamma}$ is a $Q \times Q$ matrix recomputed at each iteration. This can also be expressed by modifying (3.18) using weights,

$$\Big[\mathbf{\Gamma}^T\mathbf{W}_{\theta\varphi}\mathbf{\Gamma}\Big]_{\alpha,\beta} = \sum_{\theta,\varphi\in\Omega} w_{\theta\varphi} < \mathcal{H}\phi_\alpha(\theta,\varphi), \mathcal{H}\phi_\beta(\theta,\varphi) > \tag{3.22}$$

which needs to be recomputed at each iteration. Note again that all Hadamard distances between Hessian of basis functions are precomputed, and only the multiplication with $w_{\theta\varphi}$ and summation is performed at each IRLS iteration.



(a) $\mathcal{F}$ on image **(b)** $\mathcal{F}$ + laser points **(c)** $w_{\theta\varphi}$

Figure 3.4: Reconstructed surface $\mathcal{F}$ for a vehicle (a) projected on the image, (b) represented as a 2D depth map on discrete $\theta\varphi$ grid, and (c) corresponding weights for each grid cell. The surface is allowed to bend along patterns with low weights.

3.3.4 LiDAR Likelihood Term, $E_L(\mathbf{v})$

The expression in (3.9) represents the input data likelihood for a new laser cloud $\mathcal{P}_t$. This integral depends on surface distribution and laser data term. The laser data term $p_P(\mathcal{P}_t|\mathcal{F}_{t-1}, \mathbf{v}_t)$ encourages each new point, $\mathbf{p}_i = (\theta_i\ \varphi_i\ r_i)^T$, to fit to surface $\mathcal{F}_{t-1}$ when adjusted with $\mathbf{v}_t$. Let $\mathbf{p}'_i = (\theta'_i, \varphi'_i, r'_i)$ denote time-adjusted points,

$$\mathbf{p}'_i = \mathbf{p}_i - \Delta t_i \mathbf{R}\mathbf{v} \tag{3.23}$$

where Δt_i is the difference between t_i (time-stamp for $\mathbf{p}_i$) and $\mathcal{F}_{t-1}$ time reference. Also $\mathbf{R} = (\mathbf{r}_\theta\ \mathbf{r}_\varphi\ \mathbf{r}_\rho)^T$ is the linearized transformation at p_i that maps velocity $\mathbf{v}$ from Cartesian to spherical coordinates. Note such transformation is not in general a linear operator, and $\mathbf{R}_i$ is approximated for each $\mathbf{p}_i$. Now let $\mathbf{R}_\Omega$ be $(\mathbf{r}_\theta\ \mathbf{r}_\varphi)^T$. The energy form of laser data term is,

$$\begin{aligned} E_P(\mathbf{v}) &= \sum_{\mathbf{p_i}\in\mathcal{P}} \frac{w_i}{\tau_i^2} \left\|\mathcal{F}(\Omega'_i) - r'_i\right\|^2 \\ &= \sum_{\mathbf{p_i}\in\mathcal{P}} \frac{w_i}{\tau_i^2} \left\|\mathcal{F}(\Omega_i - \Delta t_i \mathbf{R}_\Omega \mathbf{v}) - r'_i\right\|^2 \\ &= \sum_{\mathbf{p_i}\in\mathcal{P}} \frac{w_i}{\tau_i^2} \left\| \sum_j d_j \phi_j(\Omega_i - \Delta t_i \mathbf{R}_\Omega \mathbf{v}) - r'_i \right\|^2 \end{aligned} \tag{3.24}$$

Similar to surface term, we add Huber weights w_i to eliminate outlier points. Assuming ϕ_i's are twice differentiable, and $\Delta t_i \mathbf{R}_\Omega \mathbf{v}$ is small relative to voxel size,

$$\phi_j(\theta_i', \varphi_i') \approx \phi_j(\theta_i, \varphi_i) - \Delta t_i \nabla \phi_{j,i}^T \mathbf{R}_\Omega \mathbf{v} \tag{3.25}$$

where $\nabla \phi_{j,i}$ is the gradient of ϕ_j evaluated at Ω_i. By substituting (3.25) in (3.24) and setting $r_i' = r_i - \Delta t_i \mathbf{r}_\rho^\intercal \mathbf{v}$ we get a quadratic laser energy term corresponding to a Gaussian distribution. Now let $\hat{\mathbf{d}}$ be the estimated basis function coefficients from surface reconstruction and σ_F^2 denote the surface error variance. The integral in (3.9) then results in LiDAR negative log-likelihood,

$$E_L(\mathbf{v}) = \sum_{\mathbf{p_i} \in \mathcal{P}} \frac{w_i}{\mu_i^2} \left\| \Phi_i^T \hat{\mathbf{d}} - r_i + \Delta t_i (\mathbf{r}_\rho^T - \hat{\mathbf{d}}^T \nabla \Phi_i \mathbf{R_\Omega}) \mathbf{v} \right\|^2 \tag{3.26}$$

where Φ_i is a vector whose elements are basis functions evaluated at Ω_i, and $\nabla \Phi_i$ is the matrix of stacked gradients. Also it can be shown $\mu_i^2 = \tau_i^2 + \sigma_F^2$. Also note $\mathcal{P}$ is the most recent laser scan, therefore τ_i^2 is equal to sensor characteristic range variance σ_ρ^2.

3.3.5 Image-based Data Term $E_\mathcal{I}(\mathbf{v})$

The image-based term minimizes the photometric error between appearance model $\mathcal{M}$ and new images as a function of 3D velocity $\mathbf{v}$. We model the appearance distribution in (3.8) as a Gaussian with current appearance model $\hat{\mathcal{M}}$ as mean and a pixelwise error variance of σ_M^2. After processing each new image, we update $\mathcal{M}$ as the projection of $\mathcal{F}$ on the new image.

Following the work of Kerl et al, [58], we assume the photometric error between $\mathcal{M}$ and a new image $\mathcal{I}$ follows t-distribution for each pixel. This assumption leads to non-quadratic log-likelihood which can be handled using iterative reweighting.

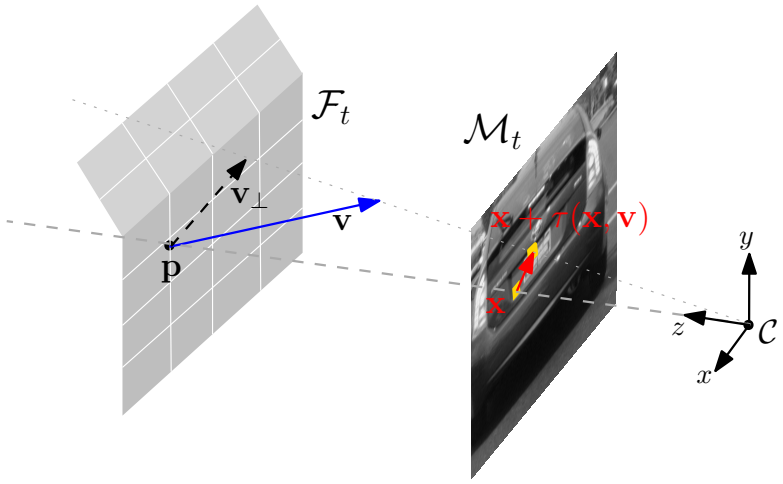


Figure 3.5: Relationship between 3D velocity vector $\mathbf{v}$ at an arbitrary point $\mathbf{p}$ on reconstructed surface $\mathcal{F}$, and the induced displacement $\tau(\mathbf{x}, \mathbf{v})$ (optical flow) on the image plane.

The energy form corresponding to $p_I(\mathcal{I}|\mathcal{M}_{t-1}, \mathbf{v}_t)$ is,

$$E_{\mathcal{I}}(\mathbf{v}) = \sum_{\mathbf{x}\in\mathcal{X}} \frac{1}{\sigma_{\mathbf{x}}^2} \|\mathcal{M}(\mathbf{x} + \tau(\mathbf{x}, \mathbf{v})) - \mathcal{I}(\mathbf{x})\|^2 \tag{3.27}$$

where $\mathcal{X}$ is the image region corresponding to $\mathcal{W}$, and $\sigma_{\mathbf{x}}^2$ denotes t-distribution weights, computed based on [58]. The effect of t-distribution weighting is conceptually similar to Huber weights, where outliers get down-weighted to have less influence in the overall estimated motion. The function $\tau(\mathbf{x}, \mathbf{v})$ denotes the pixel displacement induced by three-dimensional velocity $\mathbf{v}$,

$$\tau(\mathbf{x}, \mathbf{v}) = \Delta t \mathbf{B}_{\mathbf{x}} \mathbf{v} \tag{3.28}$$

where Δt is the time difference between appearance $\mathcal{M}$ and the new image. The 2×3 matrix $\mathbf{B}_{\mathbf{x}}$ projects 3D velocity $\mathbf{v} = (v_x \; v_y \; v_z)^T$ (in the camera reference system) onto image-space as a pixel-wise optical flow vector (see Fig. 3.5).

This matrix projects three-dimensional velocity vector $\mathbf{v} = (v_x \; v_y \; v_z)^T$ (in

the camera reference system) onto image-space as optical flow at each pixel. Let $\mathbf{p_t} = (p_x\ p_y\ p_z)^T$ denote a 3D point in camera reference system at t, and $\mathbf{x} = (x\ y)^T$ to be its image projection. Then $\mathbf{p}_{t+1} = \mathbf{p}_t + \mathbf{v}$ is the position of $\mathbf{p}_t$ at $t+1$. The induced image flow $\tau(\mathbf{x}, \mathbf{v}) = (u\ v)^T$ is given by,

$$\begin{aligned} u &= x_{t+1} - x_t = f_x \frac{p_x + v_x}{p_z + v_z} - f_x \frac{p_x}{p_z} = \frac{f_x}{p_z}\left(\frac{v_x - p_x v_z}{1 + v_z/p_z}\right) \\ v &= y_{t+1} - y_t = f_y \frac{p_y + v_y}{p_z + v_z} - f_y \frac{p_y}{p_z} = \frac{f_y}{p_z}\left(\frac{v_y - p_y v_z}{1 + v_z/p_z}\right) \end{aligned} \tag{3.29}$$

where f_x and f_y denote the horizontal and vertical focal length. Similar to Quiroga et al, [87], we apply a Taylor series for the term in denominator containing,

$$\left(\frac{1}{1 + v_z/p_z}\right) = \left(1 - \frac{v_z}{p_z} + \left(\frac{v_z}{p_z}\right)^2 - ...\right) \tag{3.30}$$

Assuming $|v_z/p_z| << 1$ only the zero-order term remains and the image flow induced by 3D motion on a pixel $\mathbf{x} = (x\ y)^T$ is given by,

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{p_z} \begin{pmatrix} f_x & 0 & -f_x x \\ 0 & f_y & -f_y y \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \tag{3.31}$$

Therefore, for an arbitrary image pixel $\mathbf{x} = (x\ y)^T$ velocity projection matrix $\mathbf{B_x}$ is given as,

$$\mathbf{B_x} = \frac{1}{p_z} \begin{pmatrix} f_x & 0 & -f_x x \\ 0 & f_y & -f_y y \end{pmatrix} \tag{3.32}$$

which depends on pixel coordinates (x, y) as well as its depth p_z. While Quiroga et al [87] employ stereo correspondences to estimate p_z, we can use the results of our intermediate depth reconstruction.

Pixel depths can be derived from reconstructed surface $\mathcal{F}$. This is where the

intermediate model connects two sensor spaces.

We convert 3D points on $\mathcal{F}$ to camera reference system, and then project them onto the image plane in order to determine pixel depths.

By linearizing $\mathcal{M}$ in (3.27) around $\mathbf{x}$ we get,

$$E_I(\mathbf{v}) = \sum_{\mathbf{x}\in\mathcal{X}} \frac{1}{\sigma_{\mathbf{x}}^2} \|\Delta t \nabla \mathcal{M}_{\mathbf{x}}^T \mathbf{B}_{\mathbf{x}} \mathbf{v} - \mathcal{M}_{t,\mathbf{x}}\|^2 \tag{3.33}$$

where $\mathcal{M}_{t,\mathbf{x}} = \mathcal{M}(\mathbf{x}) - \mathcal{I}(\mathbf{x})$ is the photometric error for $\mathbf{x}$, and $\nabla\mathcal{M}_{\mathbf{x}}$ is appearance gradient at $\mathbf{x}$. Since $\mathcal{M}$ has a Gaussian distribution, the image likelihood integral in (3.8) results in another Gaussian with log-likelihood,

$$E_C(\mathbf{v}) = \sum_{\mathbf{x}\in\mathcal{X}} \frac{1}{\sigma_{\mathbf{x}}^2 + \sigma_M^2} \|\Delta t \nabla \hat{\mathcal{M}}_{\mathbf{x}}^T \mathbf{B}_{\mathbf{x}} \mathbf{v} - \hat{\mathcal{M}}_{t,\mathbf{x}}\|^2 \tag{3.34}$$

3.3.6 Occlusion and parallax modeling

Object domain $\mathcal{W}$ determines the span of angles (θ and φ) that the object surface $\mathcal{F}$ is defined for. As shown in Fig. 3.2c, we process all object domains at the same time. This enables us to explicitly model occlusion for objects with overlapping domains for their $\mathcal{F}$. In handling both occlusion and parallax, we are merely using scene geometry from estimated $\mathcal{F}$ for all objects. For the overlapping $\theta\varphi$ cells, we mark the domain of the farther object as occluded. Another problem is that sensors at different positions lead to parallax. This means LiDAR and camera are exposed to different regions of the scene. In order to handle parallax, we use the projection of object domains onto image, i.e. $\mathcal{X}$ shown in Fig. 3.2d. In a similar way, we mark the overlapping image pixels for the farther object as occluded.

3.4 Optimization

In previous section, Camera-based and LiDAR-based energy terms on three-dimensional velocity were described. They both are quadratic functions of $\mathbf{v}$, and weighted based upon non-quadratic penalizers. As briefly mentioned before, the optimization is thus based on Iterative Re-weighted Least Squares (IRLS). Here we explain the key considerations in optimizing the objective function.

3.4.1 Inference and Tracking

The final objective function includes both Camera-based and LiDAR-based terms on velocity. These terms were ultimately described as quadratic functions of $\mathbf{v}$. Although they depend on several variables, they can be symbolically represented as,

$$\begin{aligned} E(\mathbf{v}) &= E_C(\mathbf{v}) + \alpha E_L(\mathbf{v}) \\ &= \sum_{\mathbf{x}\in\mathcal{X}} w_{\mathbf{x}} \left(\mathbf{h_x}^T \mathbf{v} - y_{\mathbf{x}}\right)^2 + \alpha \sum_{\mathbf{p_i}\in\mathcal{P}} w_i \left\|\mathbf{H_i v} - \mathbf{y_i}\right\|^2 \end{aligned} \tag{3.35}$$

where α is a weighting coefficient and tuned during testing. Note the first summation (Camera term) iterates over all pixels $\mathbf{x} \in \mathcal{X}$ and weights each term according to a pixel-dependent weight $w_{\mathbf{x}}$. This is similar to the second summation, which iterates over all points $\mathbf{p_i} \in \mathcal{P}$. We call the pair $\{\mathbf{h_x}, y_{\mathbf{x}}\}$ image-based observation for pixel $\mathbf{x}$, and the pair $\{\mathbf{H_i}, \mathbf{y_i}\}$ LiDAR-based observation for point i. Note each image-based observation provides one constraint, $\mathbf{h_x}^T\mathbf{v} \approx y_{\mathbf{x}}$, whereas LiDAR-based observations include three constraints encoded as $\mathbf{H_i}^T\mathbf{v} \approx \mathbf{y_i}$. The above energy function can be rewritten in closed form as,

$$E(\mathbf{v}) = \left\|\mathbf{\Phi}_{\mathcal{X}}\,\mathbf{v} - \eta_{\mathcal{X}}\right\|^2_{\mathbf{W}_{\mathcal{X}}} + \alpha \left\|\mathbf{\Phi}_{\mathcal{P}}\,\mathbf{v} - \eta_{\mathcal{P}}\right\|^2_{\mathbf{W}_{\mathcal{P}}} \tag{3.36}$$

where,

$$\mathbf{\Phi}_{\mathcal{X}} = \begin{pmatrix} \mathbf{h_1}^T \\ \vdots \\ \mathbf{h}_{|\mathcal{X}|}{}^T \end{pmatrix}, \qquad \mathbf{W}_{\mathcal{X}} = \begin{pmatrix} w_1 & & \\ & \ddots & \\ & & w_{|\mathcal{X}|} \end{pmatrix}, \qquad \eta_{\mathcal{X}} = \begin{pmatrix} y_1 \\ \vdots \\ y_{|\mathcal{X}|} \end{pmatrix}, \tag{3.37}$$

and $\mathbf{\Phi}_{\mathcal{P}}$, $\mathbf{W}_{\mathcal{P}}$, and $\eta_{\mathcal{P}}$ are defined in a similar column-stacked fashion. This objective function can be further simplified as,

$$E(\mathbf{v}) = \|\mathbf{\Phi}\,\mathbf{v} - \eta\|^2_{\mathbf{W}} = (\mathbf{\Phi}\,\mathbf{v} - \eta)^T \mathbf{W} (\mathbf{\Phi}\,\mathbf{v} - \eta) \tag{3.38}$$

with the new block matrices defined as,

$$\mathbf{\Phi} = \begin{pmatrix} \mathbf{\Phi}_{\mathcal{X}} \\ \alpha \mathbf{\Phi}_{\mathcal{P}} \end{pmatrix}, \qquad \mathbf{W} = \begin{pmatrix} \mathbf{W}_{\mathcal{X}} & \\ & \alpha \mathbf{W}_{\mathcal{P}} \end{pmatrix}, \qquad \eta = \begin{pmatrix} \eta_{\mathcal{X}} \\ \alpha \eta_{\mathcal{P}} \end{pmatrix}, \tag{3.39}$$

We can proceed by simply setting the derivative to zero (at each iteration) and refining $\mathbf{v}$. This results in computing an instantaneous velocity at each time instant and does not enforce temporal consistency. We follow this approach in our first experiment (involving Scala B2 LiDAR) described in Sec. 3.6.4. This leads to solving a straightforward 3×3 inverse problem,

$$(\mathbf{\Phi}^T \mathbf{W} \mathbf{\Phi})\,\mathbf{v} = \mathbf{\Phi}^T \mathbf{W} \eta \tag{3.40}$$

Although $\mathbf{\Phi}$ and η are potentially large matrices depending on number of measurements, the only matrices we need to store are $\mathbf{\Phi}^T \mathbf{W} \mathbf{\Phi}$ (3x3 matrix) and $\mathbf{\Phi}^T \mathbf{W} \eta$ (a 3-vector). This is especially important for GPU implementation (Chap. 5) where computations can be done efficiently in parallel but reading from global memory is expensive.

Maximizing the posterior distribution $p(\mathbf{v}_t|z_{1:t})$, however, requires adding a temporal consistency term. We follow this approach in our second experiment (using Velodyne HDL-64E), explained in Sec. 3.6.5. This leads to an extended Kalman Filter formulation. Let's assume our state vector is $\mathbf{v} = (v_x \ v_y \ v_z)^T$ and the evolution of state is described as,

$$\mathbf{v}_k = \mathbf{F}_k \mathbf{v}_{k-1} + \mathbf{w}_k \tag{3.41}$$

where $\mathbf{w}_k$ is distributed as $\mathcal{N}(\mathbf{0}, \mathbf{R}_k)$ and denotes process noise, and $\mathbf{F}_k$ is the state transition matrix, typically fixed to identity in constant velocity models. Then the observation model can be represented as,

$$\eta_k = \mathbf{\Phi}_k \mathbf{v} + \mathbf{u}_k \tag{3.42}$$

where $\mathbf{u}_k$ is the observation noise and distributed as $\mathcal{N}(\mathbf{0}, \mathbf{W}_k^{-1})$. The observation model depends on the number of pixels and points that belong to the object. When using a Kalman Filter, the update step requires inversion of potentially large innovation matrix $\mathbf{S} = \mathbf{\Phi}_k \mathbf{P}_{k|k-1} \mathbf{\Phi}_k^T + \mathbf{W}_k^{-1}$. This motivates us to employ the *information filter*. The main difference is we keep track of the inverse covariance (information matrix $\mathbf{Y}$) rather than covariance matrix $\mathbf{P}$. Information filter also enables us to work with 3×3 matrices rather than matrices with size equal to number of measurements (points and pixels). We keep track of both the information matrix and the measurement vector. With a constant velocity model, the prediction and update steps are mixed as,

$$\begin{aligned} \hat{\mathbf{y}}_k &= (\mathbf{I} - \mathbf{C}_k)\hat{\mathbf{y}}_{k-1} + \mathbf{\Phi}_k^T \mathbf{W}_k \eta_k \\ \hat{\mathbf{Y}}_k &= (\mathbf{I} - \mathbf{C}_k)\hat{\mathbf{Y}}_{k-1}(\mathbf{I} - \mathbf{C}_k)^T + \mathbf{C}_k \mathbf{R}_k^{-1} \mathbf{C}_k^T + \mathbf{\Phi}_k^T \mathbf{W}_k \mathbf{\Phi}_k \end{aligned} \tag{3.43}$$

where $\mathbf{C}_k = \hat{\mathbf{Y}}_{k-1}(\hat{\mathbf{Y}}_{k-1} + \mathbf{R}_k^{-1})^{-1}$. Note the last two terms for $\hat{\mathbf{Y}}_k$ and $\hat{\mathbf{y}}_k$ are the same matrices in (3.40). This formulation does not require inversion of large matrices, and allows temporal consistency through simple addition. Velocity vector and its covariance matrix can be estimated at each step as,

$$\mathbf{v}_k = \hat{\mathbf{Y}}_k^{-1}\hat{\mathbf{y}}_k, \qquad \mathbf{P}_k = \hat{\mathbf{Y}}_k^{-1} \tag{3.44}$$

This process is repeated every time a batch of measurements (pixels, points, or both) are available. An inner loop processes the all measurements at each time instant and refines robust estimation weights. This results in refined $\mathbf{W}_k$, and generates new estimates in each inner loop step. Velocity and covariance are finalized after inner loop is fully processed. In the next part we explain how robust weights are updated.

3.4.2 Non-linear Penalization

During optimization we alternate between surface and motion estimation steps, and linearize each basis function ϕ_j and all nonlinear penalizers ρ_i to get updated weights for all terms.

Before moving to the next IRLS iteration, we recompute regularization weights $w_{\theta\varphi}$ for each computing element x, e.g. pixels, points, or $\theta\varphi$ voxels. Huber weight for each x at iteration k is computed based on the residual e_x at iteration $k-1$ as,

$$w_x^k = \begin{cases} 1, & |e_x^{k-1}| < \tau \\ \tau/|e_x^{k-1}|, & |e_x^{k-1}| \geq \tau \end{cases} \tag{3.45}$$

As a consequence, outlier points, pixels and voxels will be assigned less weight over iterations; thus, $\mathbf{v}$ and $\mathbf{d}$ adapt to inliers.

3.4.3 Multi-resolution analysis

For first-order approximations in (3.25) and (3.33) to be valid, variations in $\mathbf{v}$ and $\mathbf{d}$ must be small compared to grid size. This is guaranteed by employing a coarse-to-fine and incremental approach. We start from a low-resolution grid for both $\mathcal{F}$ and $\mathcal{M}$, and continue to finer levels. Motion and surface parameters are updated at each iteration as $\mathbf{v} \leftarrow \mathbf{v} + \Delta\mathbf{v}$ and $\mathbf{d} \leftarrow \mathbf{d} + \Delta\mathbf{d}$.

We construct a Gaussian pyramid for $\mathcal{M}$, and make sure velocity variations are small relative to grid size at each pyramid level. We also employ coarse-to-fine basis functions Φ for $\mathcal{F}$. Upon retrieving new images, a Gaussian pyramid with $L = 5$ levels and down-sampling factor $\eta = 2$ is precomputed. The grid size for $\theta\varphi$ surfaces (basis function discretization rate) is also set in a similar fashion and based on image pyramid levels. The optimization algorithm starts from the coarsest level in both $\mathcal{W}$ and Ω spaces, estimating $\mathbf{v}$ and $\mathbf{d}$ (equivalently $\mathcal{S}$) up to that level. Upon convergence at level l, we move to the next (finer) image and surface level $l + 1$, and repeat the alternating optimization. This naturally encourages us to choose a multi-resolution set of basis functions, to cover the multi-resolution pyramid Ω^l. Upon moving to finer levels new basis functions are added to the set of existing ones, Φ^l, in order to complement them and to span the new level Ω^{l+1}. As we will see in Sec 3.5, B-spline wavelets are a suitable choice for hierarchical reconstruction, and they are also previously employed for dense depth extrapolation [15, 27, 59].

The pyramid implementation makes possible to compromise between accuracy and performance, analogous to histogram annealing of Held et al, [46]. If we have more processing capacity, finer levels of the pyramid are processed to attain higher accuracy.

3.5 B-Spline Wavelets as Basis Functions

Multi-resolution analysis of images and other data have proven to be very effective. Let $L^2(\mathbb{R})$ denote the set of all functions $f : \mathbb{R} \to \mathbb{R}$ for which $\|f\|^2$ is integrable over $\mathbb{R}$. A multi-resolution analysis of L^2 is a nested set of vector spaces $V^0 \subset V^1 \subset V^2 \subset \ldots$ where $clos_{L^2}(\cup_{k\in\mathbb{Z}} V^k) = L^2(\mathbb{R})$. As j increases the resolution of V^j increases. The basis functions for the spaces V^j are called scaling functions $\{\phi_i^j\}$. We define W^j as the orthogonal complement of V^j in V^{j+1}, and refer to functions spanning W^j as wavelet functions $\{\psi_i^j\}$. There are many possibilities for the choice of wavelet and scaling functions, but we prefer compactly supported cubic B-spline wavelets. Compactly supported wavelets are defined on a closed domain, e.g. $[0, 1]$. This is compatible with our set-up as we perform reconstruction for compactly supported objects too. In addition, the optimization requires continuous first and second order derivatives for convergence. B-spline wavelets of order k have $k - 1$ continuous derivatives.

Given $d \in \mathbb{Z}^+$ as wavelet order, $j \in \mathbb{Z}$ as reconstruction level, and a set of non-decreasing knots $x_0, \ldots, x_N$ with $N = 2^j + 2d$, the endpoint-interpolating (compactly supported) B-splines of degree d on $[0, 1]$ are defined recursively as follows. For $i = 0, \ldots, 2^j + d - 1$ and $r = 1, ...d$,

$$N_i^0(x) := \begin{cases} 1, & if x_i \le x < x_{i+1} \\ 0, & otherwise \end{cases} \tag{3.46}$$

$$N_i^r(x) := \frac{x - x_i}{x_{i+r} - x_i} N_i^{r-1}(x) + \frac{x_{i+r+1} - x}{x_{i+r+1} - x_{i+1}} N_{i+1}^{r-1}(x) \tag{3.47}$$

In order to get endpoint-interpolating wavelets we set the first and last $d + 1$ knots to 0 and 1, respectively. This construction gives 2^j equally spaces interior intervals, and $2^j + d$ B-spline basis functions for a particular degree d and level

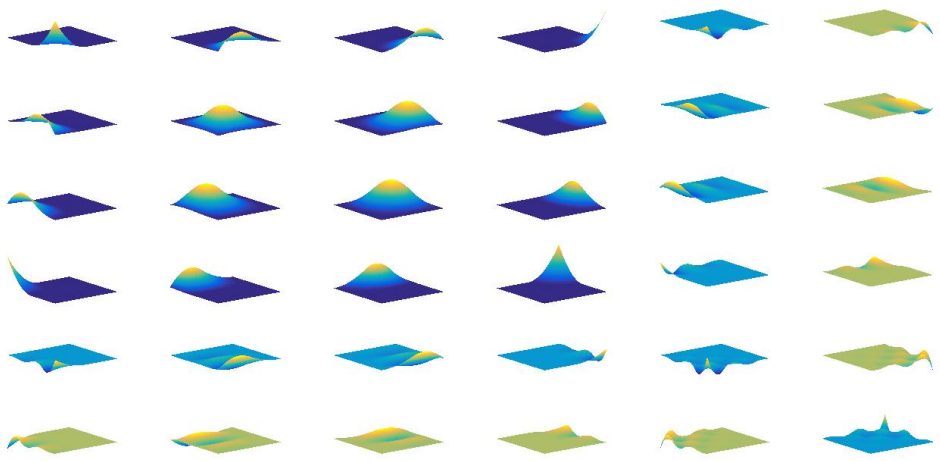


Figure 3.6: 2D B-spline wavelet and scaling functions computed for $d = 1$, $j_x = 1$, and $j_y = 1$. Top-left 4×4 block contains $\{\phi_x^j \phi_y^j\}$, top-right 4×2 block contains $\{\psi_x^j \phi_y^j\}$, bottom-left 2×4 block contains $\{\phi_x^j \psi_y^j\}$, and bottom-right 2×2 block contains $\{\psi_x^j \psi_y^j\}$

j. B-spline functions $N_0^d(x), \ldots, N_{2^j+d-1}^d(x)$ are employed in our framework as scaling functions $\{\phi_i^j\}$ that span V^j, the space of piecewise-polynomials of degree d with $d-1$ continuous derivatives. Wavelet functions $\{\psi_i^j\}$ can also be computed using refinement matrices. In order to increase multi-resolution level from j to $j+1$, we only need to add wavelet functions spanning W^j to the set of basis functions. We set $d = 3$ to get cubic B-spline wavelets with first and second order derivatives. Fig. 3.6 shows computed scaling and wavelet functions for a simple case, $d = 1$, $j_x = 1$, and $j_y = 1$.

All of the scaling and wavelet functions are precomputed, along with their numerical first and second order derivatives. As depth reconstruction is performed over a 2D grid, we need to generate 2D basis functions by simply multiplying horizontal and vertical 1D basis functions, resulting in $\{\phi_x^j \phi_y^j\}$, $\{\psi_x^j \phi_y^j\}$, $\{\phi_x^j \psi_y^j\}$, and $\{\psi_x^j \psi_y^j\}$.

3.6 Experiments

In this section we overview our sensor setup and calibration procedure, introduce our image-laser dataset for velocity estimation, present the results of the proposed method, and perform analysis and comparisons to other estimation/tracking methods. In particular, we conduct two main experiments using two different LiDARs. In the first one, we focus on the accuracy of estimated velocity vectors and do not perform tracking using Kalman or information filters. For this experiment, we use our collected dataset with Scala B2. We also test our method coupled with an information filter in order to propagate information through time. For this experiment, we employ KITTI Raw [36] and point clouds collected by HDL-64E.

In order to tune hyperparameters in the model we use 5% of sequences in each dataset for training. We need to find the optimum values for Huber function thresholds as well as step size parameters for all terms . We use Nelder-Mead to find the best set of parameters that minimize average velocity error for all objects in the training set. Since we are experimenting with different LiDARs, each dataset requires separate parameter tuning. The prototype implementation in Python takes 3.55 seconds per frame (KITTI Raw) to run on an Intel Core i7 CPU (single core).

3.6.1 Hardware setup discussion

Table 3.1 provides an overview of some LiDAR choices manufactured for automotive applications. For one of our experiments, we employ the 4-layer Valeo/IBEO ScaLa scanner. We mount it on the front bumper (at a height of 20cm) and parallel to the ground. Note most vehicles, pedestrians and traffic features are observable within its 3.2° vertical field of view. ScaLa B2 alternates between top-

Table 3.1: Comparison of four LiDAR sensors for automotive applications

	Velodyne			Valeo
	VLP-16	HDL-32E	HDL-64E	ScaLa B-2
				
Vertical FOV	30°	41.33°	26.8°	3.2°
Vertical resolution	2.0°	1.33°	0.43°	0.8°
Horizontal FOV	360°	360°	360°	145°
Horizontal resolution	0.1°-0.4°	0.1°-0.4°	0.17°	0.25°
Sampling rate	5Hz-20Hz	5Hz-20Hz	10Hz	25Hz
Distance resolution	3cm	2cm	$<$ 2cm	10cm

three and bottom-three layers, providing 3×581 range and echo-width matrices at each instance. It scans the environment at a relatively high rate of 25Hz, which makes it a suitable choice especially for highway driving. Fig. 3.7 shows some of the characteristics of ScaLa B2. We also experiment with Velodyne HDL-64E for more results. HDL-64E scans at a lower frame rate (10Hz), but provides larger vertical field of view (26.8°) and finer vertical resolution (tunable, but 0.43°). This make it appropriate for urban driving scenarios, where awareness of structure is critical and object velocities are not too high.

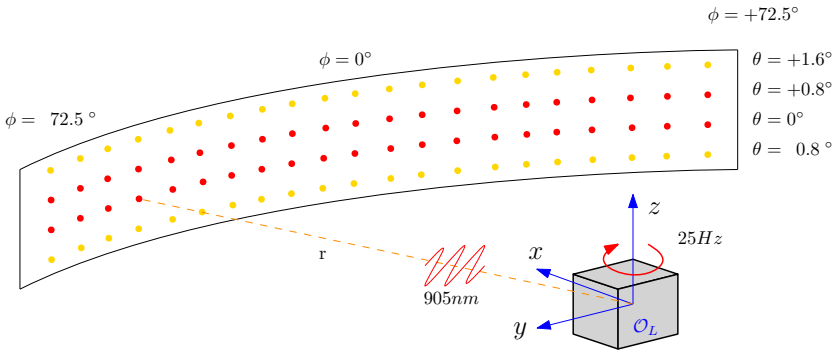


Figure 3.7: Schematic representation of Valeo/ibeo ScaLa B2 laser-scanner. Top and bottom layers (in yellow) are scanned at half frequency (12.5Hz), and middle layers (in red) are scanned at full frequency (25Hz.)

(a) Coded markers and T-shape scale bar

(b) Checkerboard in front of the car

Figure 3.8: Extrinsic calibration setup using AICON 3D System's coded stickers and scale bar.

3.6.2 Sensor Calibration

Camera intrinsics, along with lens distortion coefficients, are all computed using Bouguet's Camera Calibration Toolbox [12] for MATLAB. AICON 3D System's Digital Photogrammetry (DPA) [1] package is then employed in order to build an accurate reference system $\mathcal{V}$ and estimate extrinsic camera parameters. The photogrammetry system uses predefined coded markers and scale bars in order to accurately measure 3D objects. Once, as in Figure 3.8, stickers and scale bars are attached to vehicle body (and to a board mounted in front of it) several images are taken from various views, using a secondary hand-held DSLR camera. The photogrammetry software then post-processes the batch of images, detects all coded markers and scale bars, and constructs a coherent reference system accordingly. It proceeds with computing camera extrinsic parameters by detecting coded markers on the checkerboard (Figure 3.8b) from images taken by primary camera (mounted inside the car and behind windshield.)

We apply the benchmark method of Zhang [120] to paired camera-laser measurements of a stationary target in order to to calculate camera-to-laser system transformation, $\mathcal{T}_{\mathcal{C}\to\mathcal{L}}$.

[1] `http://aicon3d.com/products/moveinspect-technology/dpa/at-a-glance.html`

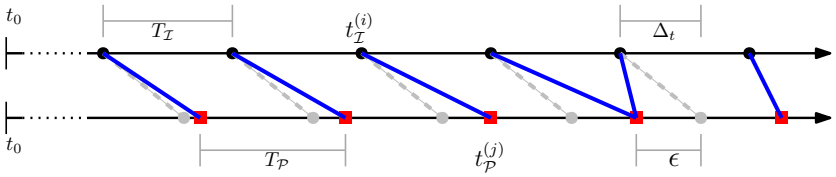


Figure 3.9: Estimation of time shift Δ_t between camera (top) and LiDAR (bottom) time references through human-provided paired measurements (blue lines). Dashed lines show image timestamps converted to LiDAR time reference.

Once calibration parameters are determined, the time shift between LiDAR scans and camera images can be estimated using a a dynamic dataset. Let $t_\mathcal{I}^{(i)}$ and $t_\mathcal{P}^{(j)}$ denote, respectively, the retrieval time of image $\mathcal{I}_i$ and average retrieval time for points in $\mathcal{P}_j$. Then,

$$\begin{aligned} t_\mathcal{I}^{(i)} &= t_0 + iT_\mathcal{I}, \\ t_\mathcal{P}^{(j)} &= t_0 + jT_\mathcal{P} + \Delta_t \end{aligned} \tag{3.48}$$

where $T_\mathcal{I}$ and $T_\mathcal{P}$ are sampling periods for camera and laser. For each $\mathcal{I}_i$, the user looks at the projection of point sets onto $\mathcal{I}_i$ and pick the best aligned match, $\mathcal{I}_j$. We, however, cannot assume if two such measurements are matched they then have the real-world retrieval time. But their time shift, ϵ, is upper-bounded by $\xi = \frac{1}{2}min\{T_\mathcal{I}, T_\mathcal{P}\}$. Figure 3.9 also pictorially demonstrates this. Then,

$$\begin{aligned} &|t_\mathcal{I}^{(i)} - t_\mathcal{P}^{(j)}| < \xi \\ &\rightarrow |iT_\mathcal{I} - jT_\mathcal{P} - \Delta_t| < \xi \\ &\rightarrow \Delta_t \sim \mathcal{U}[\Delta_{ij} - \xi, \Delta_{ij} + \xi] \end{aligned} \tag{3.49}$$

where $\Delta_{ij} = iT_\mathcal{I} - jT_\mathcal{P}$. Eq. 3.49 states that each user-provided match $\mathcal{I}_i \longleftrightarrow \mathcal{P}_j$ constraints Δ_t with a uniform distribution of length 2ξ. If the sampling rates are, even slightly, different, then aggregating more user-provided matches results in more accurate estimates of Δ_t.

Figure 3.10: Example of objects in the dataset other than vehicles

Figure 3.11: Frames 10, 20, 30, and 40 from the sample sequence

3.6.3 Evaluation metrics

In order to compare the results, we use two primary metrics to evaluate estimated velocity vectors. First we define $\|\Delta\mathbf{v}\|$ in [m/s] as the magnitude of the error vector between estimated motion $\hat{\mathbf{v}}$ and ground truth $\mathbf{v}_g$.

We also build object models by aligning the point clouds using our estimated velocity. If estimates are not accurate, the resulting accumulated point cloud will be noisy. For each object, we compute the crispness score [95] as,

$$\frac{1}{T^2}\sum_{i=1}^{T}\sum_{j=1}^{T}\frac{1}{|\mathcal{P}_i|}\sum_{\mathbf{p}_k^i\in\mathcal{P}_i} G(\mathbf{p}_k^i-\mathbf{p}_k^j) \tag{3.50}$$

where T is number of frames that object is visible, $\mathcal{P}_i$ denotes ith scanned cloud for object, and $\mathbf{p}_k^j$ denotes the nearest point in $\mathcal{P}_j$ to $\mathbf{p}_k^i$. Also, G denotes a multivariate Gaussian function. Crispness score computed above has a minimum of zero (poor reconstruction) and a maximum of one (perfect reconstruction).

3.6.4 Velocity Estimation w/o Tracking

We have implemented a simple version of our method which estimates instantaneous velocity for each object without tracking. We employ this version to assess

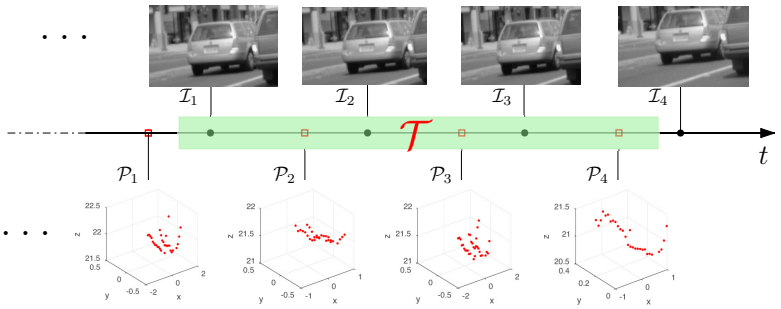


Figure 3.12: LiDAR measurements $\{\mathcal{P}_i\}$ and camera images $\{\mathcal{I}_i\}$ in a temporal window $\mathcal{T}$.

our method as a velocity estimation module. For this experiment we use 612 joint image-laser segments collected using a camera and ScaLa B2. Each segment includes five consecutive laser point clouds and five consecutive images (captured in a short time window of 200ms) of the same object. We limit the dataset to scenarios in which cars and other object are all stationary with respect to scene, but the ego-vehicle is moving. Objects in the scene, then, appear to be moving in the reverse direction of ego motion. Since we are measuring ego motion using Applanix POS LV [2], we can compute the ground-truth relative velocity of a parked vehicle in this local reference frame and quantitatively evaluate the precision of our tracking velocity estimates.

Table 4.1 summarizes the quantitative results of this experiment for various methods. ICP and KC are LiDAR only velocity estimation methods, and they work by matching each new laser point to the nearest (ICP) or all points (KC) in the 3D model $\mathcal{P}^\star$. Our ICP and KC implementations simultaneously match 5 consecutive scans in order to compute the instantaneous velocity $\mathbf{v}$. The proposed method, which employs both sensors, outperforms ICP and KC by a large margin.

The sequence in Fig. 3.11 contains some pitch angle variation as ego-vehicle is breaking. This can be observed in the blue curve in Fig. 3.13 which shows

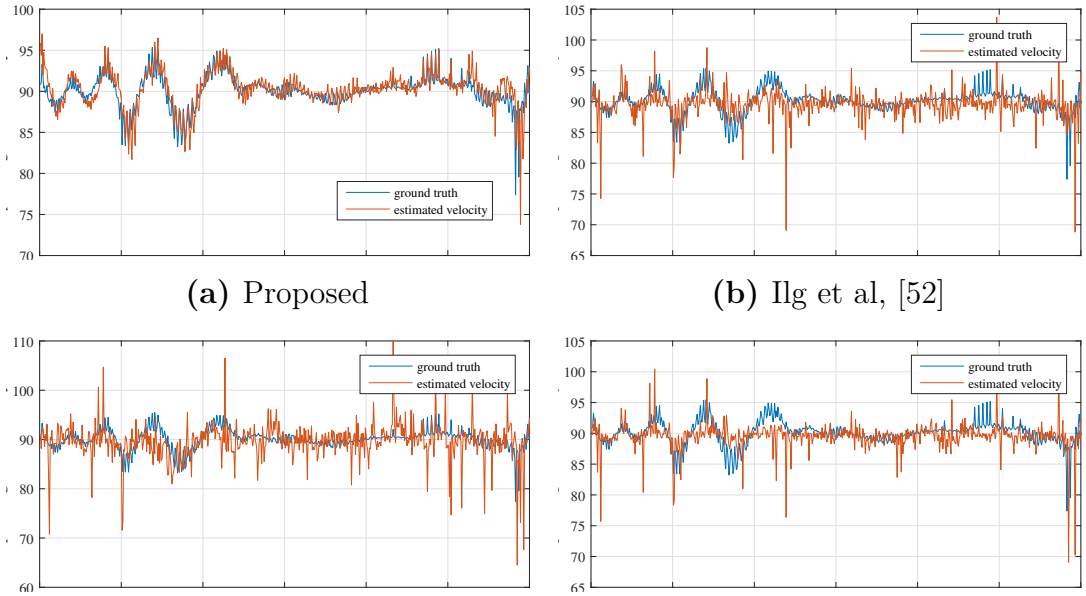
[2] `http://www.applanix.com/products/poslv.htm`

	$\|\Delta\mathbf{v}\|$[m/s]	crispness
Proposed	**0.57**	**0.18**
ICP	1.08	0.1
KC	0.89	0.13
Ilg et al, [52]	0.77	0.15

Table 3.2: Magnitude of velocity error ($\|\Delta\mathbf{v}\|$) and crispness score for different methods tested on our collected data.

pitch angle for all objects sorted by time. Note that object velocity also depends on distance – due to rotational motion component. It is also interesting to see how different methods are able to detect these variations. Needless to say pitch variations are better recovered by coupling images with laser (as in proposed method, and Ilg et al, [52]). The proposed method achieves better results primarily because image and LiDAR are processed in a tightly-coupled manner, while in the method of Ilg et al,[52] they are processed separately. They first perform dense optical flow on images, estimate a 3D motion based on pixel displacement vectors, and then perform point-to-plane ICP to get better estimates. Their method is originally geared towards denser laser scanners (in their own setup they vertically oscillate a 3-layer LiDAR to cover wider FoV), and one reason it fails here is because of the narrow FoV of our setup and poor performance of point-to-plane ICP as a consequence.

Fig. 3.14 summarizes the output of the algorithm for a real world example. Fig. 3.14(a) shows the image with green pixels indicating the projection of estimated surface $\mathcal{S}$. Fig. 3.14(b) shows aligned points as well as $\mathcal{S}$, note red points (primarily reflected from the mudguard) denote low Huber weight. Fig. 3.14(c) represents variations of $\|\mathbf{e}_v\|$ as a function of iteration, note that abrupt jumps in the curve correspond to moving to finer pyramid levels. Fig. 3.14(d-h) shows image patch in optical flow term, with red pixels indicating large matching error,



(a) Proposed **(b)** Ilg et al, [52]

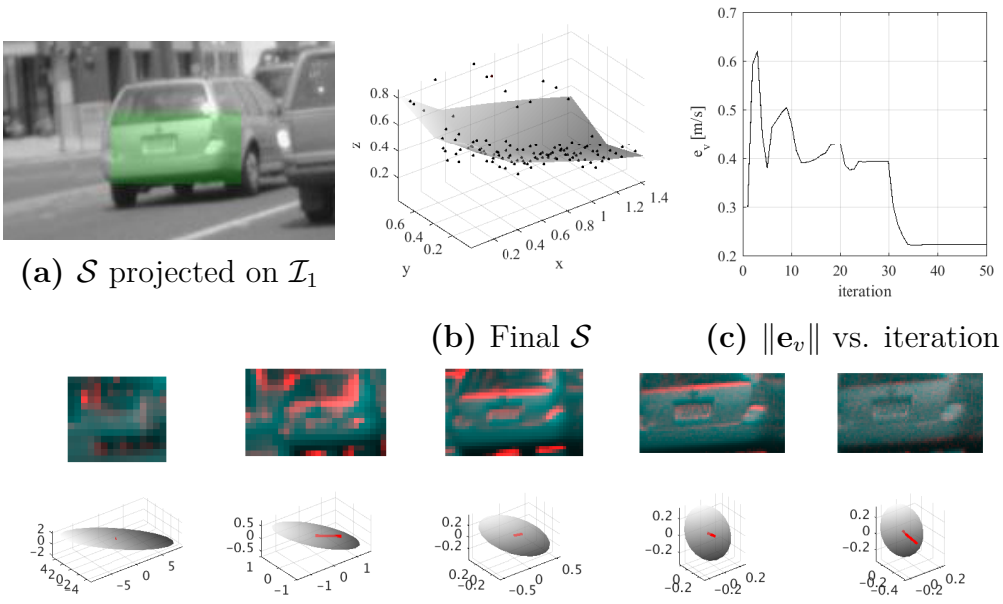
(c) 5-ICP **(d)** 5-KC

Figure 3.13: Ground truth and estimated pitch angle, θ_g and $\hat{\theta}$, for different methods over the entire sequence

as well as the evolution of covariance matrix. The ellipsoid represents 1-σ volume assigned to $\mathbf{C_v}^k$ at each iteration, and the red vector is $\mathbf{e}_v$. Note that image patches in optical flow grow in resolution with more iterations – as a consequence of moving towards finer pyramid levels. As finer pyramid levels provide more information on $\mathbf{v}$, the uncertainty ellipsoid gets smaller along xy-directions (which aligns with image grid). Fig. 3.15 also shows the results for a segment including a pedestrian.

There are also segments of which the proposed method fails to provide an accurate velocity estimate. One possibility is if optical flow term contains background pixels. This problem occurs when the image region $\mathcal{W}$ contains both target object and parts of the background. Background pixels then tend to corrupt the estimated motion, as they vote for a different velocity in the optimization process. Fig. 3.17 shows one such example. This is in fact a very common problem in

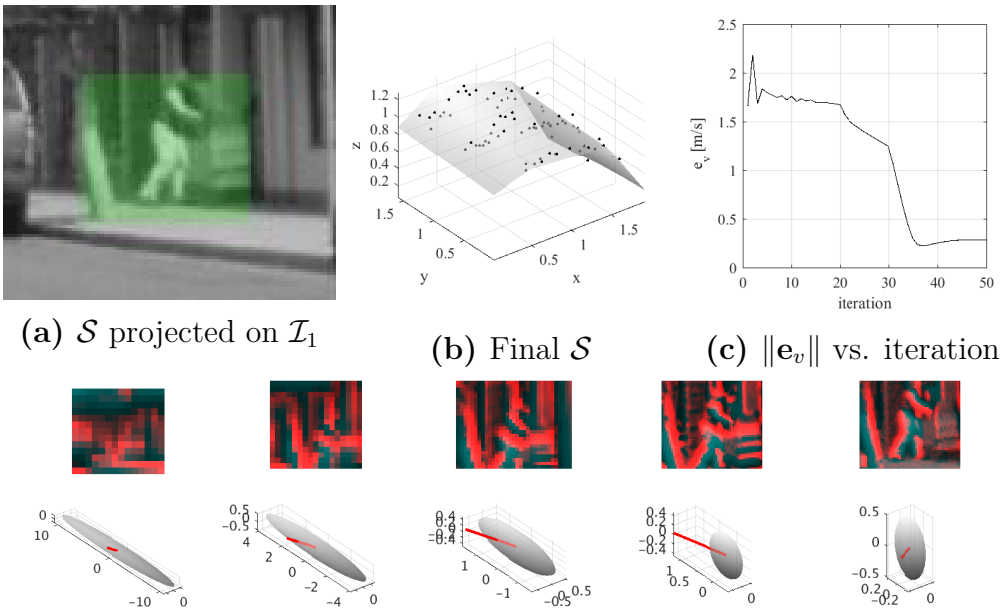
our dataset, and it only can be solved through bringing visual information into segmentation. In Chap. 4, we introduce an explicit image mask to filter out background pixels from the optical flow term. Another error source is the presence of parallax or occlusion. Fig. 3.16 shows such example, where laser points are reflected from the vehicle that barely can be seen in the camera image (the gray car behind the white one). As a consequence, the laser and image terms in the objective function are dealing with different objects. In our experiments with KITTI, parallax is explicitly modeled between all objects present in the scene. For our CUDA implementation (Chap. 5) parallax is also explicitly removed in an efficient filtering procedure.



(a) $\mathcal{S}$ projected on $\mathcal{I}_1$ **(b)** Final $\mathcal{S}$ **(c)** $\|\mathbf{e}_v\|$ vs. iteration

(d) $1-\sigma$ uncertainty ellipsoid at iterations 1, 10, 20, 30, and 40

Figure 3.14: Results of the proposed algorithm for a target vehicle. The pyramid implementation starts from a coarse-level image patch and moves to finer levels. In the bottom row, $\mathbf{e}_v^k$ is the error vector in [m/s] (in red) for iteration k, and the gray ellipsoids represents 1σ uncertainty bounds.



(a) $\mathcal{S}$ projected on $\mathcal{I}_1$ **(b)** Final $\mathcal{S}$ **(c)** $\|\mathbf{e}_v\|$ vs. iteration

(d) $1-\sigma$ uncertainty ellipsoid at iterations 1, 10, 20, 30, and 40

Figure 3.15: Results for a pedestrian. Although estimated **v** has low error, but the optical flow mask in (a) contains portions of background too.

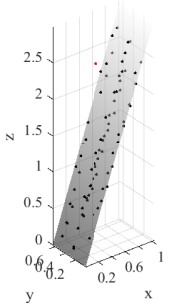
3.6.5 Velocity Estimation + Tracking

In this part we track object velocities over longer time periods. This is done by coupling our velocity estimation with an information filter. We compare our results with classic point set registration methods, e.g. ICP and Kernel Correlation (KC). To compare with multimodal trackers, we use the C++ implementation [3] of Any-time Tracker (Held et al, [46]), and implement the method of Ilg et al, [52].

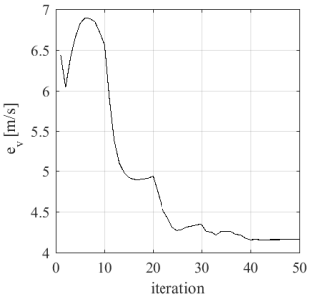
For this experiment, we use KITTI Raw [36] dataset which consists of color images and high-resolution Velodyne HDL-64E point clouds synchronized with a 10Hz sampling rate. Fig. 3.18 represents a typical driving scenario for this experiment involving two cars. As shown in Fig. 3.18, an independent model is estimated for each object. Object segments are extracted from dataset annotations

[3] `https://github.com/davheld/precision-tracking`

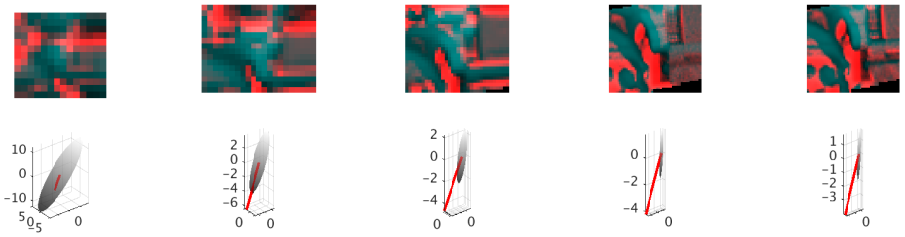
(a) $\mathcal{S}$ projected on $\mathcal{I}_1$



(b) Final $\mathcal{S}$



(c) $\|\mathbf{e}_v\|$ vs. iteration



(d) $1-\sigma$ uncertainty ellipsoid at iterations 1, 10, 20, 30, and 40

Figure 3.16: Failure due to parallax. Red regions show large matching error in image term.

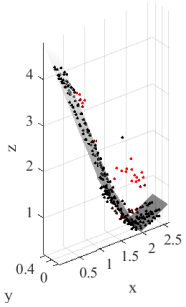
along with object ground truth velocities.

Table 3.3 shows the results for this experiment over KITTI Raw dataset. All three methods use both sensor measurements in order to track objects. Anytime Tracker (Held et al,[46]) has slightly better performance for cyclists. The primary strength of the proposed method is in tracking extended objects, e.g. cars. For objects that have negligible extent (far or small-size objects like cyclists),

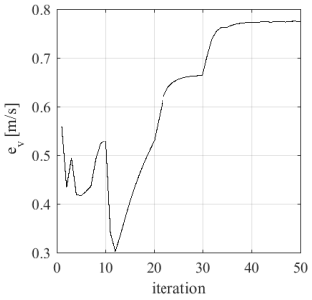
	car + van		pedestrian		cyclist	
	$\|\Delta\mathbf{v}\|$	crisp.	$\|\Delta\mathbf{v}\|$	crisp.	$\|\Delta\mathbf{v}\|$	crisp.
Proposed	**0.47**	**0.43**	**0.55**	**0.38**	0.56	0.38
ICP+Kalman	1.07	0.17	1.35	0.15	1.24	0.14
Ilg, [52]	0.88	0.28	1.05	0.23	1.04	0.25
Held, [46]	0.59	0.35	0.61	0.38	**0.55**	**0.41**

Table 3.3: Results of different methods tested on KITTI Raw [36]

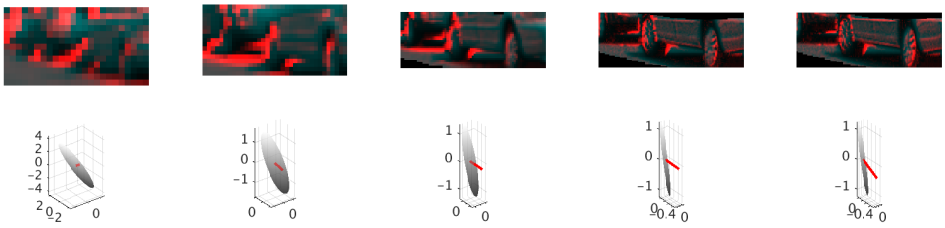
(a) $\mathcal{S}$ projected on $\mathcal{I}_1$



(b) Final $\mathcal{S}$



(c) $\|\mathbf{e}_v\|$ vs. iteration



(d) $1-\sigma$ uncertainty ellipsoid at iterations 1, 10, 20, 30, and 40

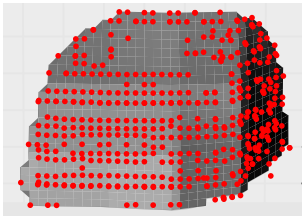
Figure 3.17: Failure due to presence of background pixels in the optical flow term.

methods which perform point-matching achieve better results. Also, Held et al, [46] achieves velocity error magnitude of 0.56[m/s] and 0.58[m/s] for objects with distance less than and more than 45m. These numbers are, respectively, 0.48[m/s] and 0.57[m/s] for the proposed method. As shown in Fig 3.18, the reconstructed surface and 3D model of near objects is richer due to more laser samples falling in each discrete $\theta\varphi$ grid cell.

Fig. 3.19 represents an example of point cloud accumulation using two different methods. When we use both LiDAR and Camera term to estimate velocity, the accumulated set of points attains a higher crispness score. When LiDAR term alone is minimized for velocity estimation the accumulated point cloud appears drifted. The last row on Fig. 3.19 plots Cartesian components of estimated velocity vector $\mathbf{v}$ for these two cases over the time window. Shaded areas represent the 1-σ uncertainty limits based upon diagonal elements of to the estimated covariance

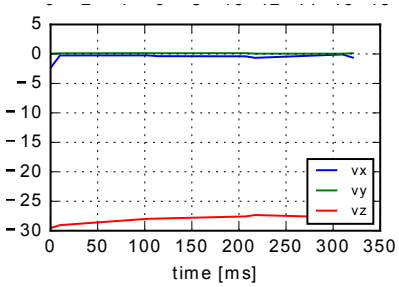
(a) Laser points projected on image. Color encodes distance.

(b) Tracked objects after 3 point clouds and 3 images.

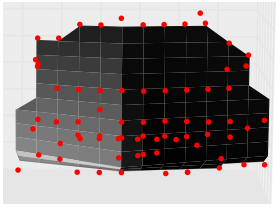


(c) left car $\mathcal{F}$

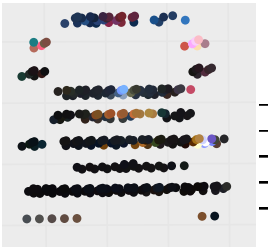
(d) left car $\mathcal{P}^\star$



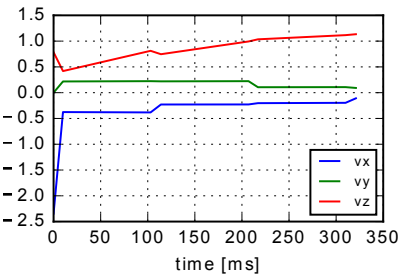
(e) **v** [m/s] vs. time



(f) right car $\mathcal{F}$



(g) right car $\mathcal{P}^\star$



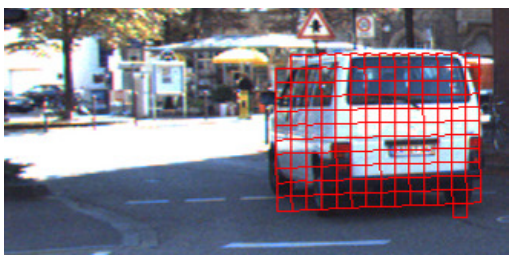
(h) **v** [m/s] vs. time

Figure 3.18: (a) shows raw LiDAR and camera measurements. (b) shows tracked cars after processing 3 point clouds and 3 images. (c-e) represent reconstructed surface, accumulated point cloud, and velocity curve for the left (near) car. (f-h) represents the results for the right (far) car.

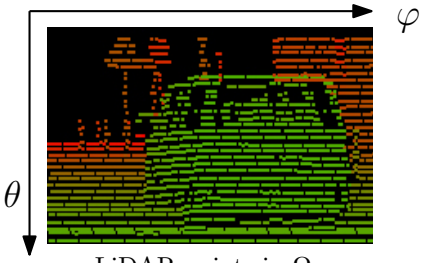
matrix.

Minimizing the image term for three-dimensional velocity estimation (or optical flow computation in general) is susceptible to unmodeled bias due to violations of brightness constancy assumption [23, 101]. Some of these violations are extreme enough that the robust estimation weights lead to down-weighting and eventually ignoring them. One such case is shown in Fig. 3.21 where a sudden shadow on object of interest results in large matching error in image-based term (shown as intensity maps). In this example, the pixels with large matching score are ignored in velocity estimation term and the bias is not present. This is evident in the error vector and 1-σ bound visualization. In several cases where the shadow is not sudden enough, it may imply illusive movement and be interpreted as a confident velocity component.

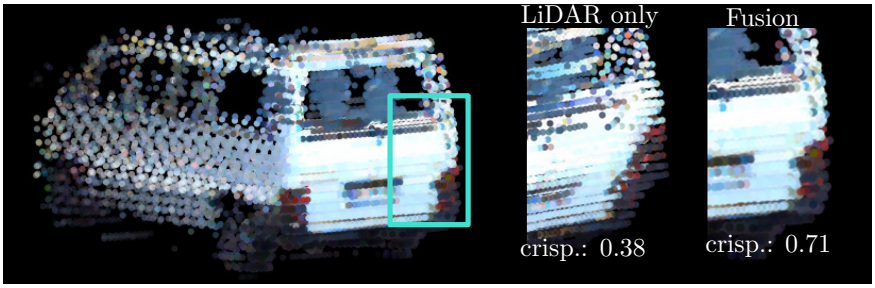
Fig. 3.20, shows a case which suffers from image-term bias due to occlusion. As long as the pixels belonging to the other vehicle are not (explicitly) excluded, a significant bias is observed in the estimated velocity after processing each image. This is evident as deviations from the ground truth (dotted line) every time the image-term is minimized. In this case, occlusions (and parallax) are modeled explicitly since all objects present in the scene are processed jointly. In our GPU implementation (Chap. 5) too, occlusion and parallax artifacts are explicitly and systematically handled by specific parallax removing cuda kernels between all neighboring tiles.



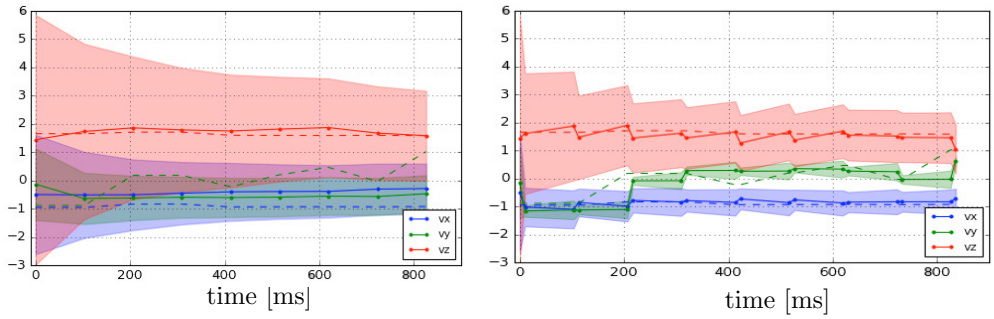
Object surface projected onto image domain



LiDAR points in Ω



Accumulated point cloud after processing 6 consecutive frames



Estimated velocity components (v_x, v_y, v_z) for LiDAR only (left) and Fusion (right) [m/s]. Shaded regions represent 1-σ uncertainty bounds

Figure 3.19: Demonstration of point accumulation and crispness score computation. Top row shows a Camera-LiDAR sample pair from KITTI. Middle row depicts the accumulated point clouds after registration, based on the estimated velocities of LiDAR-only and Fusion tracking algorithms. Last row shows velocity components and their 1-σ uncertainty bounds for these two methods.

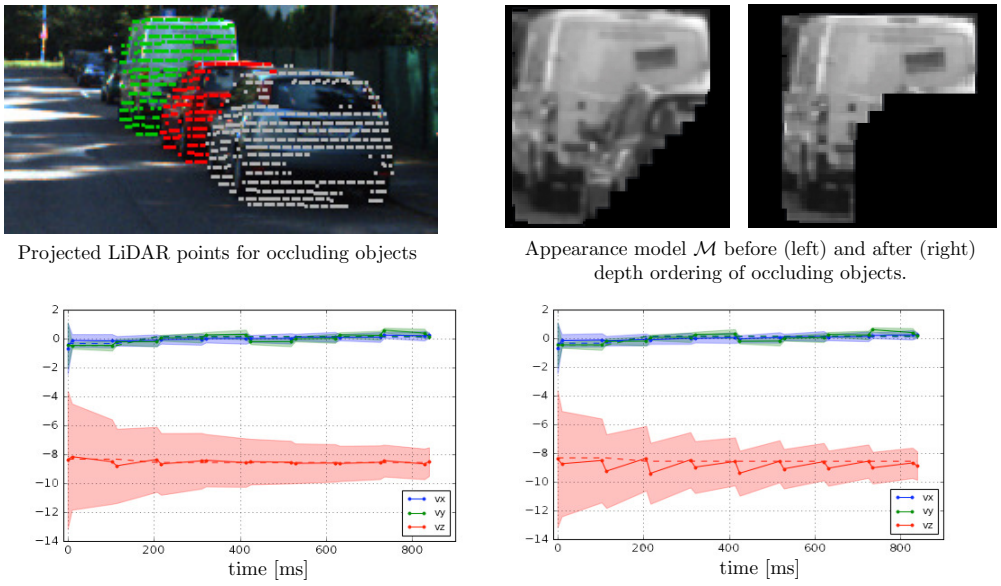


Figure 3.20: When parallax is not modeled, there is a potential for unmodeled bias when minimizing the image-term for velocity estimation.

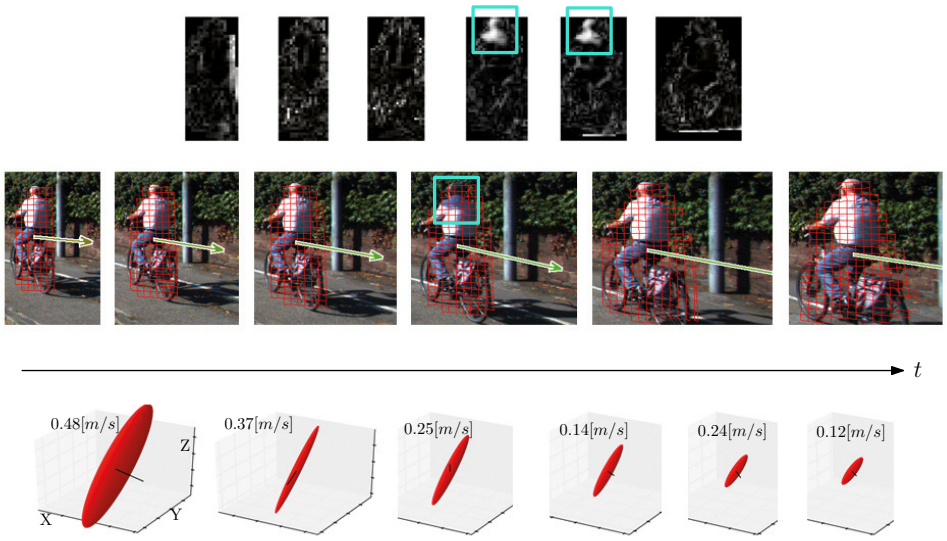


Figure 3.21: A sudden shadow on the cyclist head is tolerated in this sequence (left-to-right). Top row shows optical flow matching error, and bottom row represents velocity error and 1-σ uncertainty ellipsoids.

Chapter 4

Measurement Grouping

Modeling the 3D structure of the environment plays an important role in navigation and situation awareness. In order to understand the geometric structure of the scene, sensor measurements are usually segmented into those belonging to the ground plane and those belonging to a set of independent objects. Subsequently, a 3D model is assigned to each segment.

Over-segmenting a 2D image, i.e. superpixel segmentation, is a preprocessing technique used in many computer vision applications such as image segmentation [32] and 3D reconstruction [11]. The idea is to provide a concise image representation by grouping pixels into perceptually meaningful compact patches that adhere well to object boundaries and do not overlap with multiple objects [63]. It dramatically reduces the complexity of the subsequent task, e.g. object segmentation. For many such applications, it is far easier to merge superpixels than to split them [61].

In this work, we introduce supersurfaces which are computed by tightly-coupled processing of sparse LiDAR and dense Camera measurements. In addition to a spatial over-segmentation of the scene, supersurfaces include 3D region surfaces as well. Unlike superpixels, the goal of supersurface segmentation is to

represent possibly large surface areas of 3D objects using a parametric model. Such surfaces may be merged or tracked in the next stages, thus while preferred to be large they should not cross object boundaries. Homogeneous regions and gradients of images provide clues on object boundaries, but without depth data we are unable to reconstruct surfaces. It is possible to measure semi-dense depth in a stereoscopic vision [43] system and employ it for superpixel segmentation, but this approach requires extra processing and lacks accuracy especially in texture-less or distant areas. LiDARs as active sensors, on the other hand, give accurate range measurements at a sparse grid of azimuth-elevation angles, providing rich information about scene structure and discontinuities. Due to their sparsity, they alone cannot reliably estimate objects true boundaries. In several cases (e.g. black objects or windows in Fig. 4.1) there is remarkable missing data in LiDAR measurements which results in neglecting some object parts in segmentation. In order to combine the complementary characteristics of these sensors, we suggest to process them in a single joint Markov Random Field (MRF) energy minimization [82, 85]. This is achieved using sensor calibration parameters as well as estimated 3D surfaces that help relating sensor spaces. Fig. 4.1 summarizes our MRF-based approach for supersurface segmentation. The energy function is optimized via max-flow/min-cut and a few move-making iterations (α-expansion [13]) algorithms. The results demonstrate the adherence of extracted supersurfaces to both image and point cloud discontinuities. For our CUDA implementation in Chap. 5, we employ an over-simplified version of the grouping algorithm explained in this chapter. In particular, we allow continuous-valued masks, replace MRF with a Diffusion process, and use iterative filtering instead of α-expansion.

This chapter is organized as follows. In Sec. 4.1 earlier work on scene segmentation is studied. In Sec. 4.2 we formally state the problem. Sec. 4.3 covers

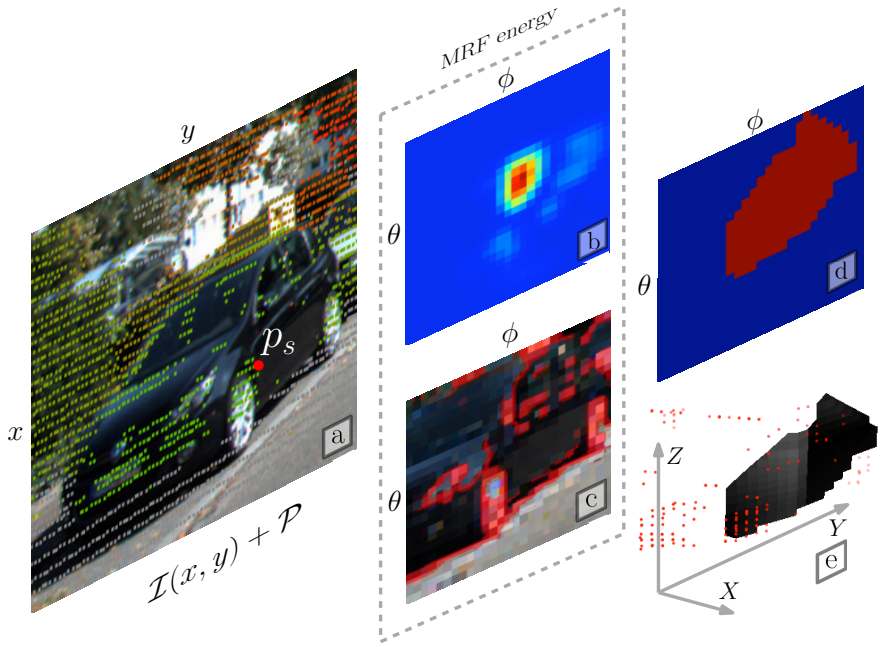


Figure 4.1: Given (a) an image $\mathcal{I}$ and a point cloud $\mathcal{P}$ and a point p_s, we compute (b) MRF unary potentials based on LiDAR points, and (c) MRF pairwise weights from image gradients on a 2D angular grid $\Omega_\epsilon = (\theta, \phi)$. The MRF solution is (d) a segment covering LiDAR points while aware of image edges. The final output is (e) a three-dimensional surface segment (or a supersurface).

our ground removing, surface reconstruction, and supersurface segmentation algorithms, and Sec. 4.4 covers experimental results.

4.1 Prior Work

As an early technique for 2D image over-segmentation, FH [29] estimates superpixels through finding minimum spanning trees of the global graph of pixels and rule-based merging of adjacent superpixels. But it does not consider compactness, size, and shape regularity of superpixels. Normalized cut [96] implicitly takes compactness into consideration, but it has limited applicability due to its high computational complexity. Turbopixel generates uniform and compact superpixels by iterative dilation of seeds that are distributed evenly. Moore et al.

[71] proposed an algorithm to preserve the topology of a regular lattice during superpixel segmentation. Liu et al. [64] suggests an objective function based on the entropy rate of a random walk and a regularization encouraging superpixels of the same size. Bergh et al. present SEEDS in [104] by encouraging color homogeneity and shape regularity in their objective function. Achanta et al. [6] proposed a simple and fast algorithm referred to as linear clustering based algorithm (SLIC). It generates superpixels by performing iterative K-means clustering in a five dimensional space combining color and spatial information. Li and Chen [63] extend SLIC to capture global as well as local similarity information. In their Linear Spectral Clustering (LSC) each pixel is mapped to a ten dimensional space in which weighted k-means approximates the spectral clustering solution in linear time. Veksler et al [107] reformulated superpixel generation as an energy minimization problem which is solved using Graph Cuts [13]. They cover the input image with overlapping square patches of fixed size, which is equal to the maximum superpixel size. Then they search for the optical assignment of pixels to patches (labels) so that the global energy is minimized. Our method is similar in spirit to their work. We also employ an energy-based approach with overlapping patches as labels, but our patches are defined as physical regions in three-dimensional space.

Superpixels have in recent years been employed for 3D scene reconstruction in stereoscopic systems [43, 116, 108]. Güney et al [43] propose using object-class specific CAD models for disparity proposal (displets) and aggregate them in a superpixel-based Conditional Random Field (CRF). Yamaguchi et al, [116] use a piece-wise planar model and segment the scene using superpixels. They use a Markov Random Field (MRF) with a static scene assumption, and manage to optimize segments, ego-motion, and plane parameters. Vogel et al, [108] find

segment plane parameters in a similar fashion, but they also consider a pairwise term between nearby planes. In order to deal with dynamic scenes, they assume a rigid motion for each segment. Menze et al, [69] group segments into objects and find motion parameters for each group, resulting in a dramatic reduction in cost.

There is another class of over-segmentation techniques that make use of RGBD data [76, 113]. Depth Adaptive Superpixels (DASP), introduced by Weikersdorfer et al [113], performs spectral graph clustering on a spatio-temporal graph of RGBD data combining position and normal information with color data. Pappon et al, [76] introduce Voxel Connectivity Cloud Superpixels (VCCS) which generates supervoxels in 3D by processing the connectivity of RGB-D depth channel. Despite their promising results on indoor RGB-D dataset, they fail to handle LiDAR point clouds of outdoor scenes especially in regions with low point density. In a probabilistic framework, Held et al, [45] improve LiDAR point cloud segmentation by adding temporal and semantic information to the conventional spatial approach. Their algorithm requires object tracking, does not take advantage of dense image data, and outputs point clusters rather than object masks.

4.2 Problem Statement

Let $\mathcal{I}(x, y)$ and $\mathcal{P} = \{p_1, p_2, \ldots, p_M\}$ denote respectively the Camera image grid and the set of LiDAR measurements. For simplicity we assume calibration parameters are given for both sensors and all measurements are synchronized to the same time reference. Note that LiDAR points of a single scan are essentially range measurements at discrete azimuth-elevation angles. This is an important point that allows us to model them simply as a sparse 2D depth map rather than a 3D point cloud. We accordingly define $\Omega_\epsilon(\theta, \phi)$ as a two-dimensional and discrete azimuth-elevation grid of cell size ϵ°. Accordingly LiDAR points are represented

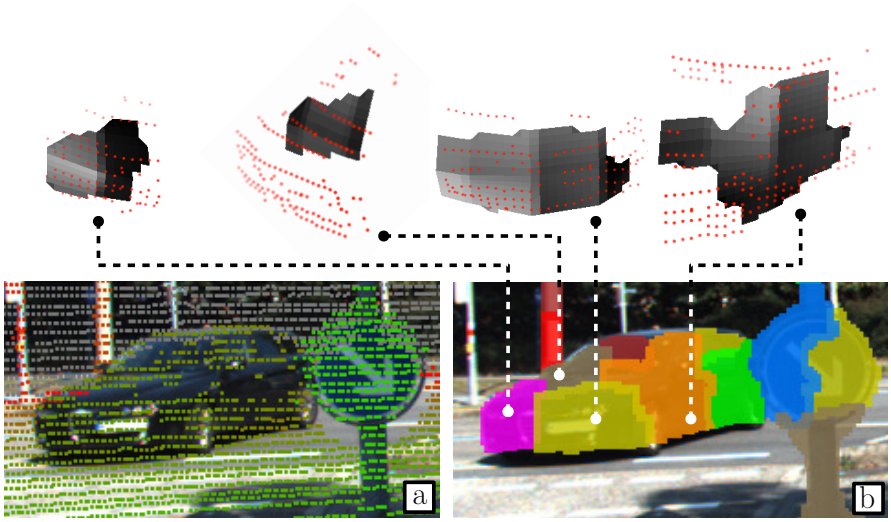


Figure 4.2: (a) shows LiDAR-image data for a black car suffering from missing data, and (b) demonstrates non-ground supersurfaces along with their 3D models.

in spherical coordinates as $p_i = (\theta\ \phi\ \rho)$ and mapped to Ω_ϵ. Now let $\mathcal{G} \subseteq \mathcal{P}$ represent the set of LiDAR reflections off the ground. We are interested in finding an over-segmentation $\mathcal{S} = \{s_1, \cdots, s_N\}$ that covers all non-ground points in Ω_ϵ. Each segment is a label mask spanning a subset of $\theta - \phi$ cells in Ω_ϵ. It is desired for segment borders to align with depth discontinuities as well as image boundaries. For each segment s is also required a surface (depth map) $\mathcal{F}_s(\theta, \phi)$ in order to capture the 3D structure of the scene.

4.3 Proposed Method

In this section we describe different components of the proposed supersurface segmentation method. The first step is to select a cell size ϵ, according to which the angular grid Ω_ϵ is defined. While smaller ϵ results in finer segment resolution, it will be less computationally efficient. In what follows we describe ground point removing, segment initialization, surface modeling for each segment, and finally first- and second-order potentials for MRF to specify object masks. Ground points

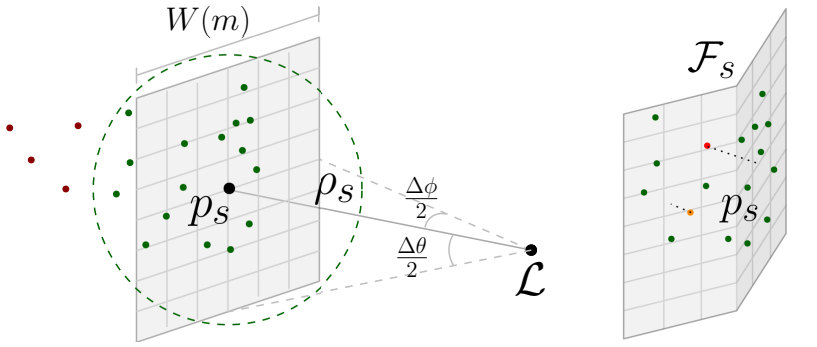
are removed in order to ensure segment masks do not overlap with the ground surface. Removing them also eliminates the connectivity of independent objects through ground. Estimating an efficient and dense depth map for each segment enables the algorithm to map surface points in Ω_ϵ to unique image pixels. This helps with connecting sensor spaces, especially for regions with missing LiDAR measurements. Fig. 4.2 shows supersurface segments computed for a car with missing data due to its color.

4.3.1 Preprocessing

As preprocessing we identify ground points $\mathcal{G}$ (along with outliers) and remove them from the point cloud. A small subset of ground points are first identified by constructing a coarse 2.5D grid for the point cloud [70]. This is followed by fitting a parametric surface model $Z = \mathcal{F}_\mathcal{G}(X, Y)$ in vehicle reference system. Then all points in $\mathcal{P}$ within a fixed threshold (10cm) of $\mathcal{F}_\mathcal{G}$ are marked as ground points. In order to handle slight curvatures, the ground surface is parametrized using the model in Sec. 4.3.3 but defined over XY plane.

4.3.2 Segment initialization

The role of this step is to initialize a set of overlapping segments that cover all non-ground points. For each segment s we set a rectangle $\mathcal{B}_f$ on the polar domain of Ω_ϵ which limits the segment area. A set of points $\mathcal{P}_s \subset \mathcal{P}$ is also assigned to each segment. Let $\mathcal{U}$ denote the dynamic set of non-ground points not assigned to any segments. The initialization process for a new segment s is to randomly choose an anchor point $p_s \in \mathcal{U}$, set $\mathcal{B}_s$ in Ω_ϵ around p_s, and assign to s the points in $\mathcal{P}$ that are within a Euclidean distance τ from p_s (Fig. 4.3a). Assigned points are added to segment point set $\mathcal{P}_s$ and removed from $\mathcal{U}$. This process is repeated



(a) $\mathcal{B}_s$ in LiDAR space **(b)** Estimated $\mathcal{F}_s$

Figure 4.3: Each segment s has (a) a set of assigned points (in green) and a square box $\mathcal{B}_s$ centered on its anchor p_s, then (b) a 3D surface $\mathcal{F}_s$ is fit to assigned points.

until $\mathcal{U} = \emptyset$. Note that assignment to multiple segments is possible for points, as we do not remove them from $\mathcal{P}$. The associated rectangle $\mathcal{B}_s$ (as shown in Fig. 4.3a) is in Cartesian space a 3D rectangle centered on p_s and normal to the line connecting LiDAR origin $\mathcal{L}$ to p_s. We constrain them to be a square of size W meters, and compute the corresponding size on Ω_ϵ as,

$$\Delta\phi = \Delta\theta = 2tan^{-1}\left(\frac{W}{2\rho_s}\right) \tag{4.1}$$

where ρ_s is the range (depth) for p_s. Fig. 4.4b-c represents as an example the estimated $\mathcal{B}$s for all segments extracted from a sample scene. Note that W has a physical meaning and can be set based on expected object size and regardless of object position. This is in contrast to the method of Veksler et al [107] where maximum segment size is defined in terms of pixels. If available, semantic information for p_s can be employed to specify W.

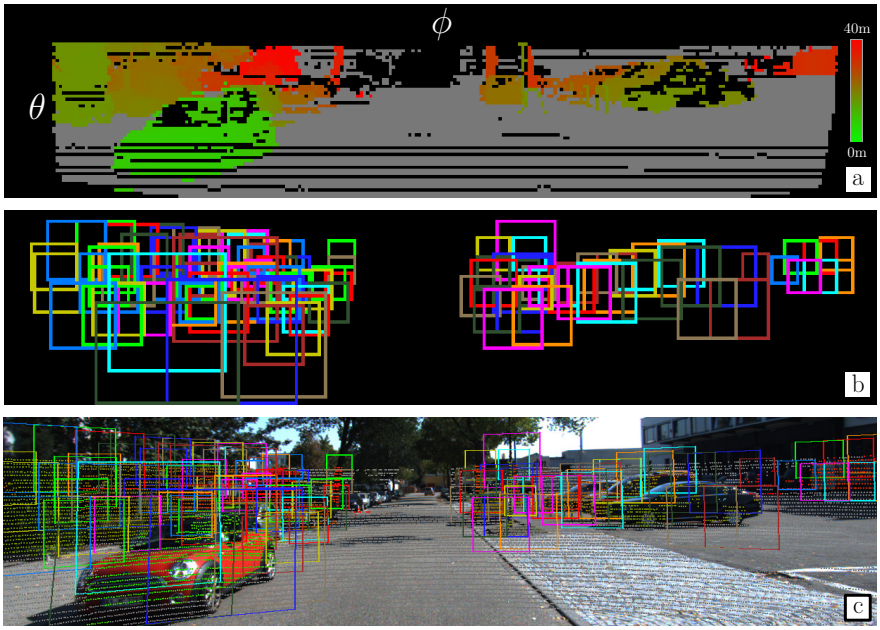


Figure 4.4: (a) LiDAR points on $\Omega_\epsilon = (\theta, \phi)$ grid with gray representing ground (or far) points, (b) square boxes $\mathcal{B}$ for all segments in Ω_ϵ, and (c) projection of points and boxes on image.

4.3.3 Surface reconstruction

In order to associate each (θ, ϕ) cell in $\mathcal{B}_s$ to its corresponding image pixel, we need to have an estimated depth for that particular cell. Since points in $\mathcal{P}_s$ are sparse (especially in regions with missing LiDAR data) and do not cover all cells we need to fit a dense surface and interpolate depth. Segment surface $\mathcal{F}_s$, defined as a depth map over $\mathcal{B}_s$ grid, is estimated through minimizing an energy term consisting of a data and a regularization term. The data term encourages $\mathcal{F}_s$ to align with points in $\mathcal{P}_s$. We also add a non-quadratic (Huber [49]) penalizer to handle outliers. Similar to [116, 108], we also add a piecewise planar prior term. The prior term helps with depth interpolation in regions where LiDAR measurements are not available, e.g. the black car in Fig. 4.2. Inspired by the regularization term in [59, 15], we minimize the second-order derivatives of $\mathcal{F}$ to encourage smooth surfaces. In addition, we want the surface to potentially

fold along sparse line patterns to prevent over-smoothing. To accomplish this, we add a spatial Huber penalizer to the regularization term as well [22]. Assuming $p_i = (\theta_i\ \phi_i\ \rho_i)$, surface energy term is defined as,

$$E_{\mathcal{F}} = \sum_{p_i \in \mathcal{P}_s} \rho\Big(\mathcal{F}(\theta_i, \phi_i) - \rho_i)\Big) + \gamma \sum_{\theta,\phi \in \mathcal{B}_s} \rho\Big(\|\mathcal{H}\mathcal{F}(\theta, \phi)\|_F\Big) \tag{4.2}$$

where γ is a constant, $\rho(.)$ denotes Huber penalizer [49], $\|.\|_F$ is the Frobenius norm, and $\mathcal{H}$ represents Hessian matrix. The energy term in (4.2) is minimized using Iterative Reweighted Least Squares (IRLS). For more efficiency, we represent $\mathcal{F}_s$ as a linear combination of N_B basis functions,

$$\mathcal{F}(\theta,\ \phi) = \sum_{j=1}^{N_B} d_j \phi_j(\theta,\ \phi) \tag{4.3}$$

Basis functions employed in our algorithm are introduced in Sec. 4.4.1. At each IRLS iteration, the data term will be a quadratic function of $\mathbf{d} = (d_1, \cdots, d_{N_B})$. As shown in Klowsky et al [59], the regularization term can also be reformulated as a quadratic function of $\mathbf{d}$. Huber weights for data term decrease during re-weighting iterations for outliers (encoded with color in Fig. 4.3b). In the regularization term, most (θ, ϕ) cells will be assigned large Huber weights, resulting in penalization of second-order derivatives and smooth surface. But these weights will decrease for a sparse pattern of cells, which leads to less regularization and possibly surface bending.

4.3.4 MRF-based mask optimization

Through reconstruction approach of Sec. 4.3.3 we have a dense and piece-wise smooth surface computed over maximum surface rectangle $\mathcal{B}_s$. The purpose of this

section is to find the optimal mask on $\mathcal{B}_s$ that cuts segment surface $\mathcal{F}_s$ against other segments as well as ground. Finding the optimal supersurface mask is defined as an energy minimization on an MRF over Ω_ϵ. An MRF formulation helps to aggregate LiDAR-based and image-based information about segment mask. Since we have multiple segments with overlapping $\mathcal{B}_s$, a multi-label MRF is employed. We find the optimal labels ℓ for (θ, ϕ) cells that minimize the MRF energy. Let $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s)$ define a graph for each segment. $\mathcal{V}_s$ is the set of (θ, ϕ) cells on $\mathcal{B}_s$ and $\mathcal{E}_s$ is the set of 8-neighborhood edges in $\mathcal{B}_s$ connecting nearby cells. We define MRF unary (first order) potentials $D(i)$ for each vertex v_i based on geometric features in LiDAR space, and compliment it with pairwise (second-order) potentials $V(i, j)$ for each edge $\{v_i, v_j\}$ based on image gradients. The unary potential describes the likelihood of a particular (θ, ϕ) cell to be occupied by segment s. While the pairwise term enforces shape smoothness and encourages boundaries to coincide with sharp image edges. Let ℓ be the global labeling for all cells in all segments. The overall energy for segment s given ℓ is,

$$E_s(\ell) = \sum_{v_i in \mathcal{V}_s} [\ell_i = s] D(i) + \sum_{\{v_i, v_j\} \in \mathcal{E}_s} [\ell_i = s, \ell_j \neq s] V(i, j) \tag{4.4}$$

where $[\cdots]$ is Iverson bracket, and ℓ_i denotes the segment label for v_i. The global MRF energy is defined simply as,

$$E(\ell) = \sum_{s \in \mathcal{S}} E_s(\ell) \tag{4.5}$$

where $\mathcal{S}$ is the set of all segments including an extra ground segment. As we have a multi-label energy we must use move-making approximation algorithms [107], e.g. $\alpha\beta$-swap or α-expansion. In most cases it is sufficient to perform 3 iterations of α-expansion [13] to converge to a locally optimal solution. In what follows we

describe in detail the computation of $D(i)$ and $V(i,j)$.

4.3.5 First-order potentials

The unary term for segment s is the combination of a prior D_p and a data term D_d. The prior assigns larger potentials to cells that are closer to the anchor $p_s = (\theta_s\, \phi_s\, \rho_s)$. And the data term is defined in terms of the projection of segment assigned points $\mathcal{P}_s$ onto $\mathcal{B}_s$. Represented as a spatial grid, the overall unary term is computed as,

$$D(\theta,\phi) = G_s(\theta,\phi) + \alpha \sum_{p_i \in \mathcal{P}_s} w_i\, K_i(\theta,\phi) \tag{4.6}$$

where G_s and K_s are 2D Gaussian kernels, α is a constant, and w_i is the weight assigned to p_i. G_s has mean (θ_s, ϕ_s) and standard deviation $\Delta\theta/6 = \Delta\phi/6$, and covers the whole box. K_i is centered on (θ_i, ϕ_i) and has a variance equal to LiDAR's characteristic angular error variance. Point weights w_i are also computed as,

$$w_i = \exp\Big(\frac{-d(p_i,p_s)^2}{2\tau^2} + \frac{-(\rho_i - \mathcal{F}(\theta_i,\phi_i))^2}{2\sigma_F^2}\Big) \tag{4.7}$$

where $d(.,.)$ denotes Euclidean distnace, τ is the assignment threshold from Sec. 4.3.2, and $\sigma_F = 0.3m$. Note the weighting function in (4.7) gives less weight to surface outliers or points that are far from the anchor.

4.3.6 Pairwise potentials

Let $\Gamma(v)$ denote the projection of a vertex v onto the image. For each $\{v_i, v_j\} \in \mathcal{E}_s$, the idea behind pairwise potentials is to see if the line segment connecting $\Gamma(v_i)$ and $\Gamma(v_j)$ crosses any strong gradients. This approach has been successfully employed as an affinity metric in contour-based segmentation [8, 7]. Similar to [8] we define the affinity distance $d_e(v_i, v_j)$ as the maximum edge map value along the

connecting line segment. The pairwise energy between v_i and v_j is then defined as,

$$V(i,j) = \xi \exp\Big(\frac{-d_e(v_i,v_j)^2}{2\sigma_e^2}\Big) \tag{4.8}$$

where σ_e and ξ are constants. Note if the line segment crosses an edge the pairwise potential $V(i,j)$ will be close to zero. This encourages v_i and v_j to have different labels. We also project the vertex 3D point $\mathbf{X} = (\theta\ \phi\ \mathcal{F}_s(\theta,\phi))$ as,

$$\tilde{\mathbf{x}} = \mathbf{K}_c\ [\,\mathbf{R}\,|\,\mathbf{t}\,]\ \mathbf{X} \tag{4.9}$$

where $\tilde{\mathbf{x}}$ is the projected point in homogeneous coordinates, $\mathbf{K}_c$ is camera projection matrix, and $[\,\mathbf{R}\,|\,\mathbf{t}\,]$ is the extrinsic parameters converting LiDAR to camera reference system. We simply use Canny edge detector to quickly compute image edge map.

4.4 Experimental Results

In order to evaluate our method, we test it on 30 LiDAR-image samples from KITTI raw dataset [36]. LiDAR point clouds have 64 vertical layers and are synchronized with image data. Using Interactive Segmentation Tool [1], object contours are annotated on the image plane for this subset. Estimated supersurfaces are projected onto the image plane (Sec. 4.3.4) and their boundaries are compared with ground truth. We use average boundary recall (Avg BR) to evaluate the results of various methods. BR measures average fraction of ground truth edges falling within at least $2\epsilon^\circ$ of a segment boundary. High BR shows that the superpixels are well aligned with ground truth object boundaries. Since we are interested in

[1] `https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html`

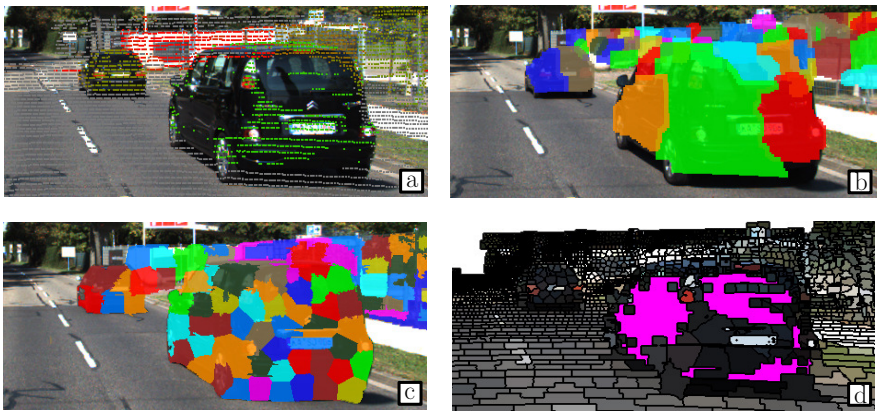


Figure 4.5: Superpixels generated for (a) a sample scene using different methods, (b) proposed method, (c) SLIC [6], (d) DASP [113]. No superpixel is generated for the pink region.

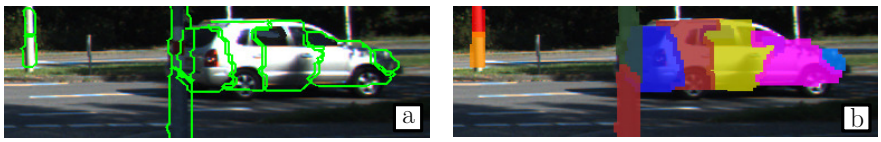


Figure 4.6: Transferring supersurfaces from Ω_ϵ to image plane is susceptible to parallax, as the pole occludes the car in (a). This can be solved through (b) depth-ordering of surface segments.

efficient representation of the scene, there is a preference for fewer segments for a fixed BR. Therefore, we tune the segment size parameter for each method so they produce relatively the same number of segments to cover non-ground regions.

4.4.1 Implementation

We set maximum supersurface size W to 2.0 meters and angular cell size ϵ to 0.2° for our experiments. Model parameters are selected based on LiDAR range and angular accuracy, and the constants are tuned heuristically. Depending on ϵ we need different sets of nested basis functions to cover the angular grid Ω_ϵ. Let $L^2(\mathbb{R})$ denote the set of all functions $f : \mathcal{R} \to \mathcal{R}$ for which $\|f\|^2$ is integrable over $\mathcal{R}$. A multi-resolution analysis of L^2 is a nested set of vector spaces

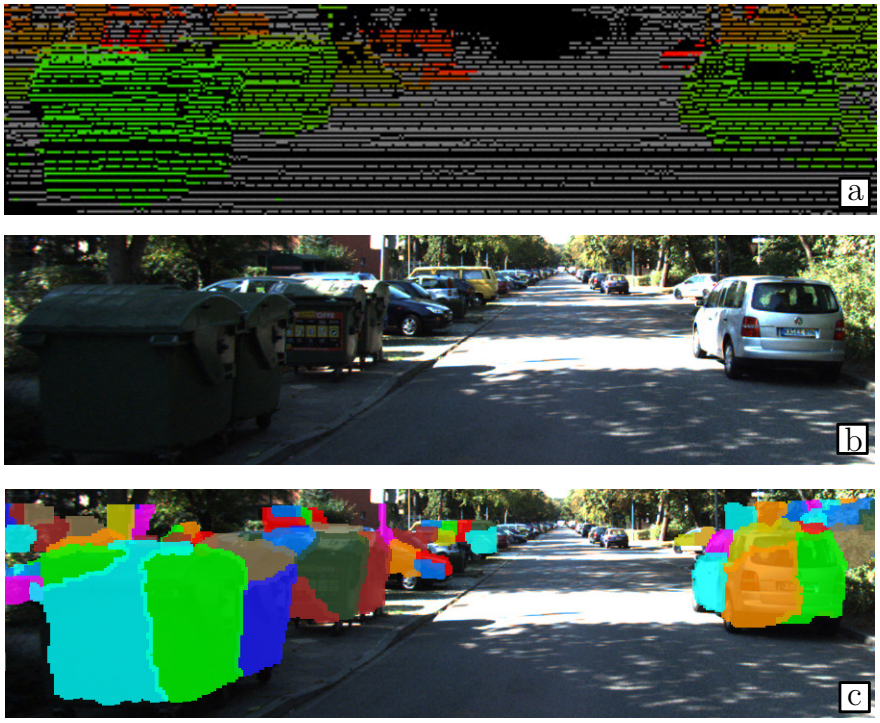


Figure 4.7: A sample scene from KITTI containing obstacles of different classes, with (a) LiDAR measurements and (b) image data. Supersurface segments are extracted for non-ground points in (c).

$V^0 \subset V^1 \subset V^2 \subset \ldots$ where $clos_{L^2}(\cup_{k\in\mathbb{Z}} V^k) = L^2(\mathbb{R})$. The basis functions for the spaces V^j are called scaling functions $\{\phi_i^j\}$. There are many possibilities for the choice of multi-resolution basis functions, but we choose square B-spline wavelets [15] due to their simplicity and continuous first and second order derivatives. Note these derivatives (used to compute $\mathcal{H}$ in Eq. 4.2) are pre-computed and cached. Currently the method is implemented in Python and uses `pymaxflow`[2] to implement segment graphs and to perform max-flow/min-cut. With this implementation and selected parameters it takes 5.5 seconds on average to process one frame on a single-threaded Intel Core i7 CPU.

[2] `https://github.com/pmneila/PyMaxflow`

4.4.2 Discussion

We compare our supersurfaces with the results of SLIC [3] [6] and Veksler et al [4] [107] as two image-based superpixel segmentation techniques. We also experiment with DASP [5] [113] as an RGB-D based method. Since DASP requires a depth channel with the same size as the input image, we project our sparse point clouds onto the image plane and generate semi-dense 16-bit `pgm` files. Fig. 4.5 shows the results different methods on a scenario consisting of two cars at different distances. Depth-aware techniques (supersurface and DASP) produce segments with approximately similar physical size. While image-based methods (SLIC) produces segments that are of the same spatial size. As shown in Fig. 4.5c, this results in significantly more segments for near object while they have a fixed physical size. This is a disadvantage for 3D scene processing, as we prefer efficient object representations for tasks such as tracking. There is also noticeable missing data in LiDAR measurements of Fig. 4.5a due to vehicles dark color. As a result, DASP fails to generate segments for these regions (marked as pink in Fig. 4.5d). Table 4.1 summarizes the quantitative results for various methods in terms of average boundary recall (Avg BR) for a relatively fixed average number of segments covering non-ground regions. Superpixel segmentation techniques that are purely image-based (SLIC and Veksler) have poor boundary recall when lower number of segments are allowed (or equivalently segment size is large). This is primarily because image boundaries do not in general coincide with object boundaries, and with larger segments a smaller fraction of image boundaries are covered. DASP performs better compared to SLIC and Veksler, but has less boundary recall compared to the proposed method. This is partly due to its

[3] `http://ivrl.epfl.ch/research/superpixels`
[4] `http://www.csd.uwo.ca/~olga/Projects/superpixels.html`
[5] `https://github.com/Danvil/asp`

	SLIC [6]	Veksler [107]	DASP [113]	Proposed
Avg Seg	73.6	77.5	71.1	70.2
Avg BR	0.66	0.62	0.75	0.83

Table 4.1: Average boundary recall (Avg BR) and average number of segments for non-ground regions (Avg Seg) for two image-based (SLIC, Veksler), one RGB-based (DASP), and the proposed method.

failure to estimate surface normals from simulated `pgm` depth map.

4.4.3 Parallax and class-independence

Since LiDAR and camera are mounted at different positions on ego-vehicle, they have different viewpoints and observe different things. This is referred to as parallax, and results in overlapping segments when they are transferred from Ω_ϵ to the image plane. Fig. 4.6a represents such case where pole segments are occluding the vehicle. As shown in Fig. 4.6b, this can be resolved by depth-ordering of layers and checking their estimated surface $\mathcal{F}_s$.

Fig. 4.7a-b represents a scene with irregular obstacles from different classes with ground LiDAR points marked in gray. Segmented supersurfaces shown in Fig. 4.7c are computed for non-ground regions, and work equally well for all objects regardless of their class.

Chapter 5

GPU Implementation

This chapter describes a real-time adaptation of different pieces of the proposed method assembled together. Two previous chapters were devoted to the explanation of our velocity estimation (3) and measurement grouping (4). Here they are assembled in a simple integration which is suitable for Graphical Processing Units (GPU). In particular, we talk about a CUDA implementation of the proposed method which takes into account the structure and limitations of most GPUs. We leverage the power of Streaming Multiprocessors (SM) and the structure of thread blocks and processing threads along with their limitations in shared memory. Some of the proposed algorithms in previous chapters are fundamentally changed here in order to suit the hardware specifications. In the end, our CUDA implementation comes out as a one piece algorithm that successfully borrows and implements the main conclusions of previous chapters.

After going over a list of algorithm modifications, we introduce *fusion planes* and *overlapping tiles* as two building blocks of this CUDA implementation. Then CUDA kernels employed are briefly explained depending on their level: plane level, tile level, and inter-tile level. Fig. 5.1 simply represents the data structures employed in our simplified CUDA implementation.

5.1 Practical Adaptations

This section covers the main modifications applied to our velocity estimation and measurement grouping methods before they are integrable as a one-piece CUDA implementation.

The processing plane (referred to as *fusion plane* in this chapter) is chosen to lie on the image domain. This is as opposed to the approach in Chap. 3 where fusion took place on Ω, LiDAR's Spherical system. This is mainly to eliminate the need for Jacobian matrix computation between two modalities. As another advantage, surfaces do not need to be projected onto the image plane since they live there now. Only LiDAR points need to be projected onto image plane – which is computationally less expensive. Thus, this modification improves performance with almost no sacrifice. Needless to say, we apply the multi-resolution processing principle and construct a pyramid of fusion planes. Unlike Chap. 3, the fusion plane pyramid has the same resolution as image Gaussian pyramid.

Object-level (or segment-level) surface reconstruction is also replaced with a plane-level depth-map extrapolation. As shown in Fig. 5.1, each fusion plane has multiple grids of variables including a depth-map. In Chap. 3 each segment was modeled by a dense and continuous surface. This leads to an inverse problem, and the underlying basis function coefficients were iteratively estimated. Our motivation behind such model was to handle dark objects and their missing data issues. Here and after projecting LiDAR points onto image plane, we extrapolate depth using data adaptive kernels while estimating the underlying uncertainty in interpolated values. Although this approach does not handle missing points in the cloud, due to its filtering nature it is appropriate for CUDA integration. In addition, the problem of missed measurements due to unreflective surfaces is being solved at the hardware level in the next generations of LiDAR [4].

The most fundamental change is the introduction of a grid of overlapping patches covering each fusion plane. Fig. 5.1 represents such grid as part of each fusion plane. These patches have a fixed size of 16×16, they are referred to as *tiles*, and they are equivalent to supersurfaces. In Chap. 4 supersurfaces were introduced to fit inside an arbitrary square with fixed physical dimensions. These frames could be initialized anywhere on the image plane, and could theoretically have any pixel-wise dimensions. In this chapter, we have a predefined set of possibilities for such supersurfaces. As an advantage, supersurfaces are no longer irregularly structured and they lie on a predefined grid with known connections. This simplifies any further inter-tile processing, since the neighbors for each tile are known according to the grid. Fixed-size tiles are inspired from the concept of thread blocks in CUDA. Thread blocks are a combination of warps (or threads) that have access to the same shared memory. Shared memory helps eliminate multiple redundant readings from global memory, and if properly used significantly improves performance. This makes thread blocks of size 256 a suitable choice to process each tile. Although tiles are of fixed size 16×16 on each fusion plane, they have different apparent dimensions depending on which level of the fusion pyramid they belong to. Tile are also chosen to be overlapping in order to accommodate the possibility of overlapping supersurfaces. The overlapped region also helps in identifying connectivity of neighboring tiles. Each tile has a fixed-size mask that represents the area occupied by the present object, as well as a set of edge coefficients that describe the connectivity to the nearby tiles. Similar to Chap. 4, each tile has an independent velocity vector and Covariance matrix that describe the motion of all pixels and points within the tile mask.

There are also slight modifications in some of the sub-components; for example, ground plane elimination, LiDAR-term in velocity estimation, and also

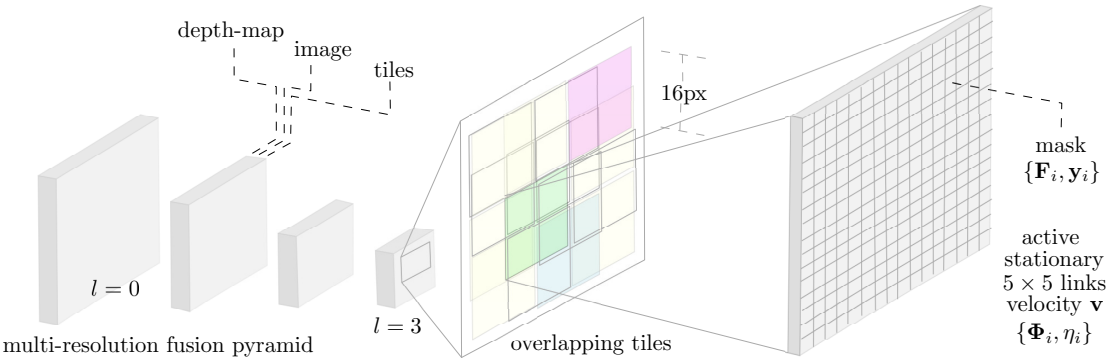


Figure 5.1: Schematic representation of fusion planes in the multi-resolution pyramid, image plane and depth-map at each level, grid of structured overlapping tiles, and tile-level variables in each grid.

α-expansion for solving the multi-label MRF for binary mask segmenation. Instead of fitting a piece-wise planar surface to model ground, here we mark ground points based on geometric constraints of nearby points on LiDAR Spherical grid. The LiDAR term in velocity estimation is also slightly modified to make computations faster. Finally, the MRF of Chap. 4 is simplified through allowing continuous valued tile masks. This makes it possible to employ diffusion process and a few iterations of data-adaptive filtering instead of computationally expensive graph cuts or α-expansion. Also, filtering is by far more suitable for parallel processing in CUDA.

5.2 CUDA Kernels

We divide the discussion on our CUDA implementation based on the functionality of each kernel. Some early kernels perform operations at plane level mostly as preprocessing, e.g. generating image pyramid, point cloud projection, and depth extrapolation. Another set of kernels operate on tile level and constitute the segment-wise procedures like mask propagation and measurement computation. And some other operations involve looking at a subgraph of nearby tiles,

Figure 5.2: Multi-resolution image pyramid generated in preprocessing

for example computing the connectivity of tiles or even velocity estimation. In order to estimate velocity for each tile (segment) measurements from nearby tiles (if they are connected) are also aggregated before the underlying linear system is solved. This section briefly explains the key points in each kernel.

5.2.1 Plane-level kernels

A 4-level pyramid of planes is employed as the main space for data fusion. Each of these planes, referred to as *fusion plane*, is employed for regions within a specific range. Along the same lines of the discussion in Chap. 2, closer measurements (points and pixels) are processed at coarser levels of this pyramid while farther measurements are processed at finer levels. We choose these fusion planes to lie on image plane in order to eliminate cross-reference Jacobian computations of Chap. 3. After some local processing, LiDAR points are also projected onto this plane and employed to extrapolate depth.

Gaussian image pyramid kernel creates a multi-resolution pyramid of the original image through consecutive anti-aliasing low-pass filtering and down-sampling. We compute a total of 6 levels, and computations are done in parallel with 32×32 thread blocks. An example of different pyramid levels is shown in Fig. 5.2. Since the lowpass filter has window size of 5 pixels, an apron of 2 pixels is considered in each block. This results in $28^2/32^2 \approx 0.76$ efficiency in parallelization.

We have more levels in the image pyramid than fusion pyramid. Scene reconstruction and velocity estimation only happens at 4 levels and in the fusion pyramid. However, velocity estimation and optical flow in general require lower-resolution images. This is due to the Taylor approximation in the brightness constancy objective, and in order to be valid requires small motion vector. By starting from lower resolution images (and depth-maps) the Taylor approximation is reasonably valid. In our implementation, fusion planes are processed with down-sampling factors of 1, 2, or 4, depending on matching error. We refer to this as *processing level* as opposed to fusion level.

Local point cloud processing, projection, and ground removal is actually a LiDAR grid level kernel. With Velodyne HDL-64E the LiDAR measurement space is a 64×2048 grid of range values ρ in meters. For each cell θ (radians) and φ (radians) are known as well. We employ the calibration parameters to compute the image coordinates of LiDAR point projections. This is done very efficiently using 32×32 thread blocks with no apron. Then in another kernel with apron of size 1, the connectivity of each LiDAR point to its 8-neighbors is measured and encoded in an 8-bit integer. These variables combined with heuristics are also employed to mark some LiDAR points as ground. LiDAR points that are not marked as ground and are within FOV of the fusion pyramid are used to fill the corresponding pixels in the (zeroth level of) fusion depth-map. The result is (in most cases including KITTI) a sparsely filled depth map. Projected points onto the fusion plane are also susceptible to parallax.

Removing parallax is the next kernel with 32×32 thread blocks, vertical apron of 5 and horizontal apron of 2. For each pixel (not on aprons) a thread looks at a 5×11 window center on it to find potential parallax. We remove the mid-point if a closer points (exceeding some threshold) exists in the window. The difference

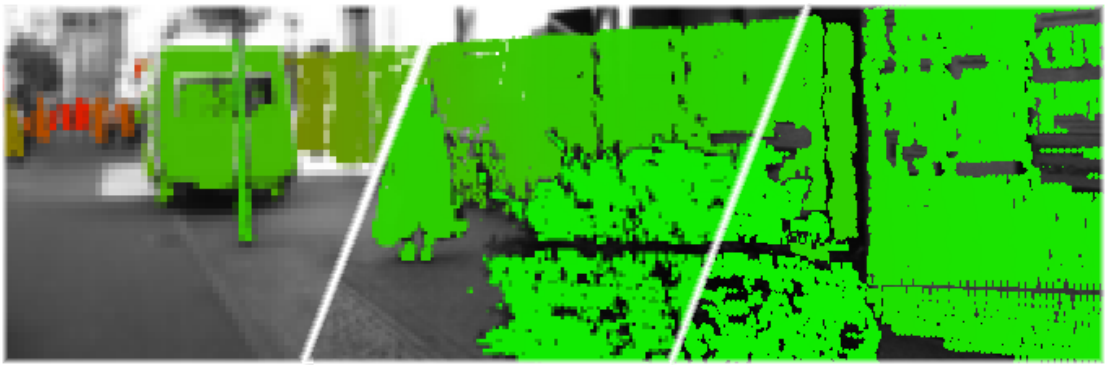


Figure 5.3: Reconstructed surface on different fusion planes

between vertical and horizontal window size used is due to the difference between horizontal and vertical angular resolutions of LiDAR. In order to reduce global memory readings the sparse projections are initially stored on shared memory.

Depth map extrapolation is done using 16×32 thread blocks processing the sparse depth-map. Sparse range values are stored on shared memory, along with the 8-bit integers describing their local directional connectivity. The directional weights are coupled with decaying radial weights to construct a data-adaptive weight w_i for each neighbor cell. Each cell is then processed with an independent thread which performs weighted averaging of all N points that fall inside the 5×11 window. Numerator is computed as $w_i \rho_i$ and denominator as w_i, both summed over $i \in \{1, ..., N\}$. Pixel range is then estimated by dividing numerator by denominator, and denominator is regarded as a measure of confidence.

Depth down-sampling kernel is then applied three times, in order to consecutively fill in the values of range and confidence for coarser levels of fusion pyramid depth-maps. This is equivalent to Gaussian image pyramid generation, but aliasing cannot be eliminated through lowpass filtering since they generate smoothing artifacts in depth-map. Thread blocks of size 32×32 load range and confidence values and fill the corresponding 16×16 square in the lower-level plane. Thus, each 2×2 square produces a pair of range-confidence. If the values inside a

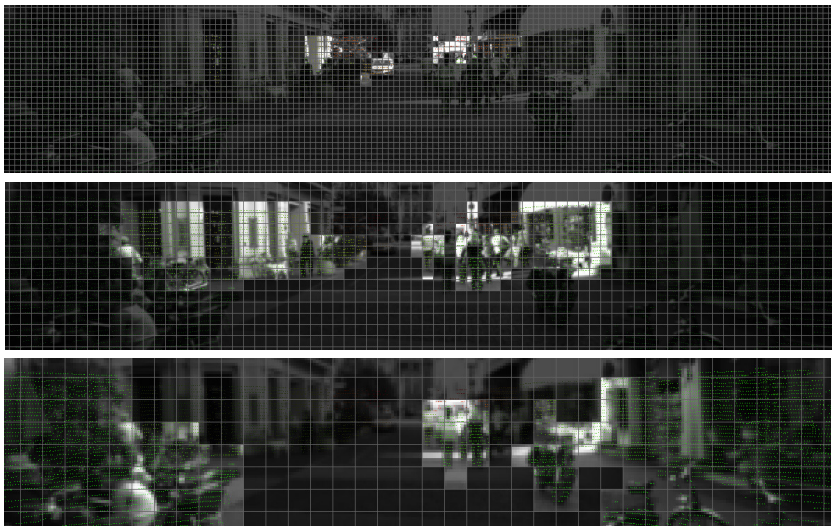


Figure 5.4: Multi-resolution overlapping tiles employed at different levels for processing objects of different distance. Dark tiles are inactive due to low accumulated non-ground point confidence.

square suggest the presence of a depth edge (high variance) then the corresponding confidence is decreased. Range is also computed as the average of non-occluded cells weighted by their confidence. This procedure results in low (or even zero) confidence if the range variance is too large. Fig. 5.3 shows an example of upsampled depth-map for finest level (right) and the next two coarser levels (middle and left). Uncertainty in estimated depth is visualized as transparency channel.

5.2.2 Tile-level kernels

These kernels operate at tile-level, have thread blocks with the same size as tiles, and compute necessary variables inside each tile. With our 4-level fusion pyramid, measurements are assigned to the appropriate level based upon their range. If upper threshold for zeroth level range is set to be τ_0 then lth level has a range threshold of $2^l\tau_0$. These tile-level kernels are then applied to every tile on every level of the fusion pyramid.

Tile activation kernel prunes those tiles that fail to attain a minimum con-

fidence. Confidence values from depth-map (corresponding level) are summed through logarithmic reduction and compared with a threshold. This is done using 16×16 overlapping thread blocks, each covering one tile. Fig. 5.4 shows an example with three levels of the fusion pyramid. Active tiles are brighter than inactive tiles. Although finer levels have more tiles, but a small fraction of them are activated in this case. In general, this depends on the distribution of object distances in the scene.

Seed initialization and mask generation kernels are performed on active tiles in order to estimate their occupied region. A seed point is computed per tile as one of the high confidence pixels which has minimum range (closest to sensor). The mask pixel that corresponds to the seed is initialized with a large value, and an iterative region-growing method is applied next. This is equivalent to the α-expansion step of Chap. 4, but replaced with a direct filtering-based region-growing. Filter weights are computed according to (confidence-aware) range and intensity similarity between neighboring pixels. The resulting mask is non-binary, but values are encoded as 8-bit integers to reduce global memory readings/writings. Also, each tile mask is down-sampled twice to generate an 8×8 and a 4×4 mask. These masks are required during observation computation with different *processing levels*. Fig. 5.5 shows masks (in red) from connected tiles in the proximity of a selected tile (in purple). Tile connections are computed in inter-tile kernels. Note that masks in (b,c,e) are from a coarser fusion plane than (d,f). The observation matrices and vectors from pixels in red are weighted in order to compute tile velocity. Each red region effectively represents a connected component of the scene.

Mask normalization is necessary in order to prevent potential overcounting of measurements. All pixels on each fusion plane belong to 4 overlapping tiles,

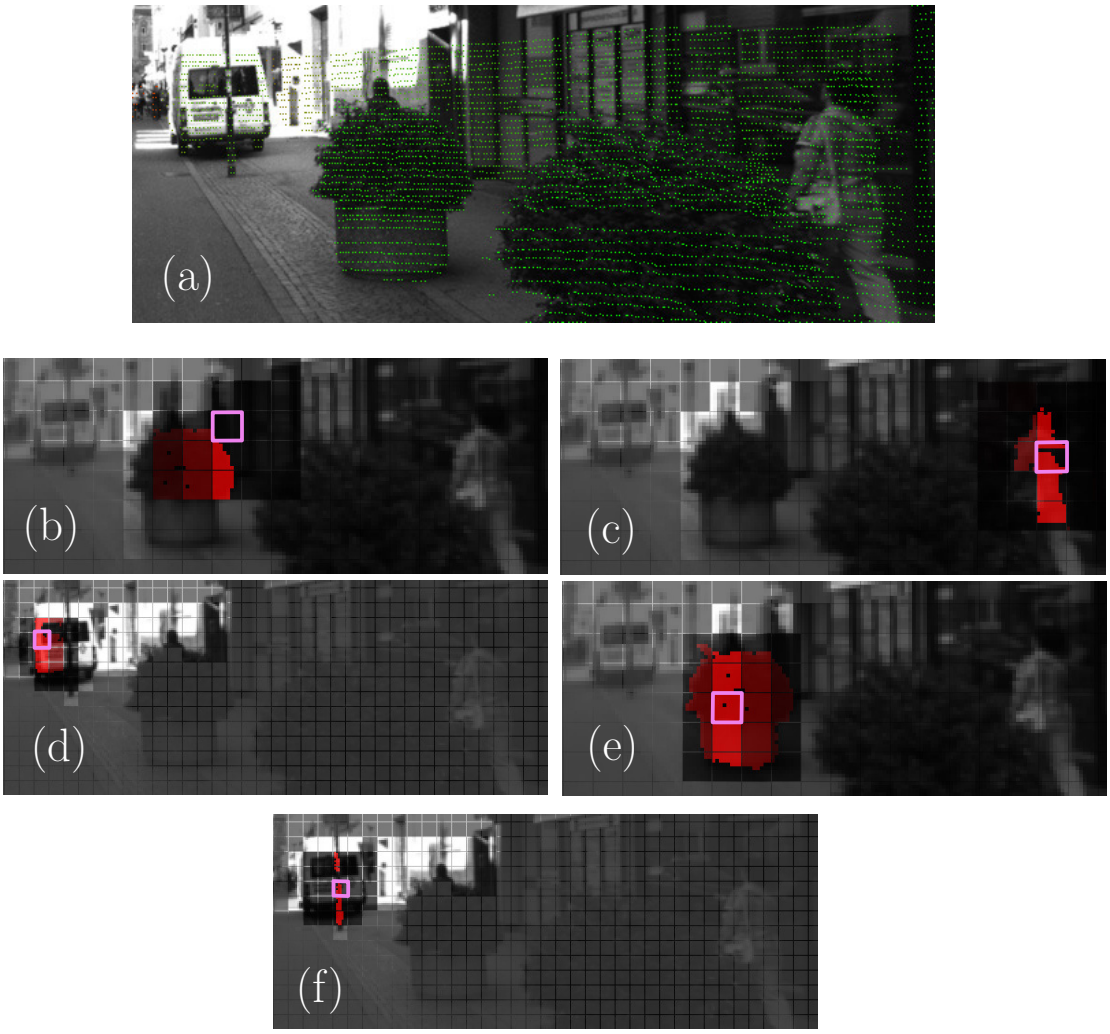


Figure 5.5: Examples of tile masks. In each case the main tile is marked in purple, and masks from connected tiles in the 5×5 proximity are also shown. Redness of each pixel encodes the underlying coefficient by which the underlying observations are weighted.

and the sum of corresponding mask variables may be larger than one. This is the case, for example, when all tiles cover the same object. This leads to over-confident estimates of object velocity. Mask normalization both handles such cases and also computes the overlapped area of neighboring tile masks. Overlapped areas are later employed in computation of tile connectivity.

Observation computation is the core CUDA kernel that implements LiDAR-based and Camera-based (optical flow) quadratic terms for velocity estimation. Each pixel generates one image-based and three point-cloud-based observations on three-dimensional $\mathbf{v}$ of the tile. This can be represented in compact form as $\mathbf{F_i v} = \mathbf{y_i}$ where the pair $\{\mathbf{F_i}, \mathbf{y_i}\}$ denotes the observation parameters of pixel i. Each thread compute one observation pair, and they are weighted and averaged in a reduction step to generate tile-level observation, $\{\mathbf{\Phi}, \eta\}$.

This kernel initially employs coarse-level masks, and moves to the next level after inner-loop iterations. Non-quadratic robust weights (Huber or t-distribution) are refined after each iteration – resulting in modified mask weights.

5.2.3 Inter-tile kernels

These kernels are applied to all planes in the fusion pyramid, with each thread processing one tile. In this implementation, tile-level proximity is set to as a 5×5 window centered around each tile.

Inter-tile connections are computed prior to observation aggregation. In the 5×5 proximity around each tile, connectivity weights are estimated based upon Euclidean distance of tile seed points, overlapped areas computed in mask normalization kernel, and inter-tile robust estimation. We also employ the heuristics in the geometric nature of the problem; for instance, we are more confident on connectivity of tiles that are placed vertically than horizontally. Inter-tile robust

estimation weights are refined after each velocity estimation iteration, and they depend on the Mahalanobis distance of estimated velocity vectors.

Observation aggregation and velocity computation follow after observation and inter-tile connectivity computation. Here observations $\{\mathbf{\Phi}, \eta\}$ from nearby tiles (5×5 window $\mathcal{W}$) are linearly combined according to the inter-tile connections. Fig. 5.6 symbolically shows the aggregation step. This is equivalent to edge-preserving smoothing effects of data-adaptive kernels like Bilateral [100] filter. Intuitively, this step encourages *connected* tiles to have the same three-dimensional velocity. After reduction-based linear combination of observations, the resulting 3×3 linear system is solved simply using Cramer's rule. Then tile velocities are updated on global memory.

Fig. 5.7 depicts examples of velocity estimation on different fusion planes. Color hue encodes motion direction on xz plane (in Camera reference system), and saturation encodes magnitude. In our CUDA implementation, we employ IMU measurements as part of KITTI dataset to remove the effect of ego-motion. This is a straightforward and linear procedure, but it depends on tile distance since ego-motion has rotational component too. Unlike Chap. 3, estimated velocities here are thus relative to the ground reference.

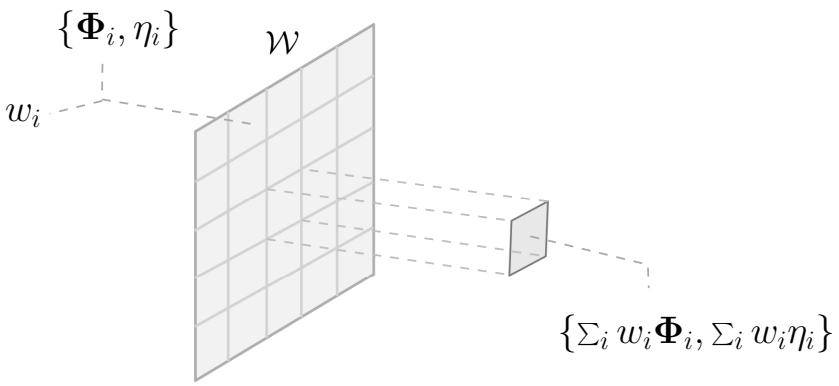


Figure 5.6: Aggregating tile-level observation matrices $\mathbf{\Phi}_i$ and observation vectors η_i weighted by connectivity coefficients w_i between middle and neighboring tiles in a 5×5 proximity.

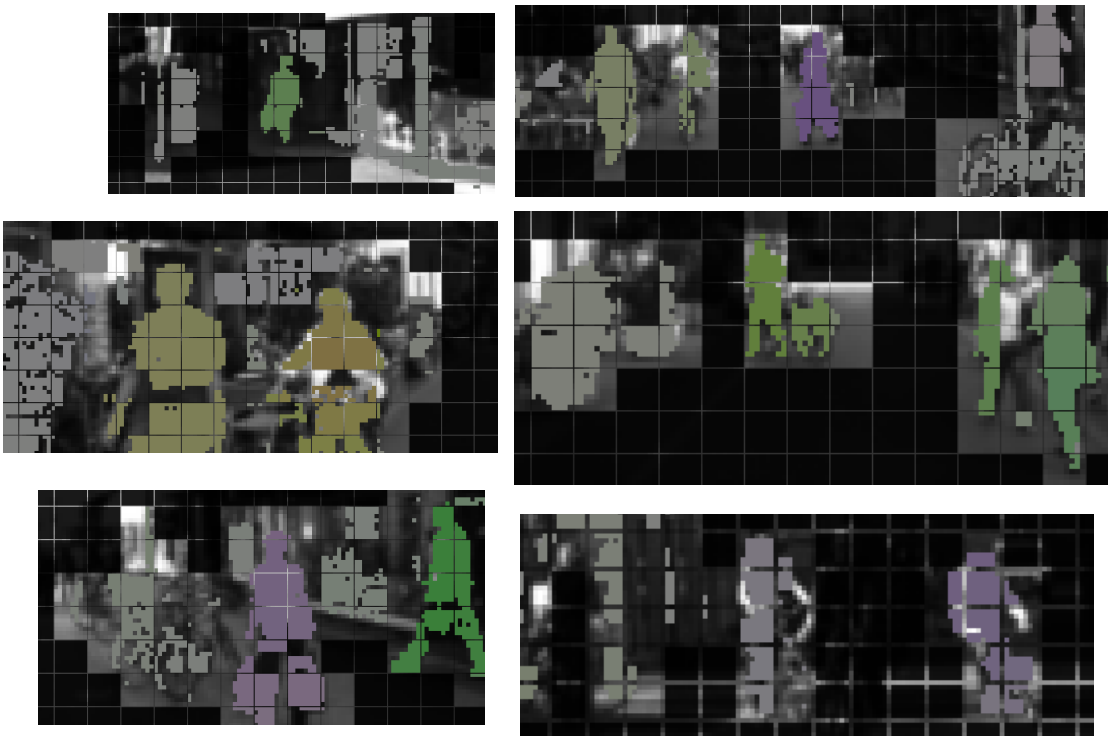


Figure 5.7: Examples of object masks, with hue encoding direction and saturation encoding magnitude of estimated velocity with respect to ground.

Chapter 6

Conclusion

This dissertation was devoted to the study of joint processing of range and intensity measurements to perform simultaneous geometry and motion reconstruction for highly dynamic urban driving scenes. In Chap. 2 the problem was formally stated and our motivation behind it was explained. Inspired by recent developments in early-vision algorithms (optical flow [14, 117], stereoscopic [116, 108, 60] or RGB-D [87] scene flow) we strive towards dense computation of three-dimensional velocity vectors without the need for object semantic classification. This makes the tracking module independent of instance-level detection or object identification. It also eliminates error due to poor instance localization, missed detections, or presence of objects of unknown class. High-level vs. low-level fusion paradigms were also described, and it was explained why we are interested in low-level (measurement-level or early-stage) fusion of data. Despite extensive efforts in LiDAR-based [46] or multi-sensor [52] tracking, our proposed method is the first joint LiDAR-Camera fusion approach to multi-object tracking and scene reconstruction. We also studied the nature of each measurement space, and managed to describe the limitations involved in early-stage fusion. In order to attain estimation errors lower than a desired threshold, we showed that objects

should be processed at different resolutions depending on their distance. A multi-resolution fusion grid was then employed, so that closer objects are processed at coarser resolutions. This multi-resolution framework is a natural choice to balance between performance and accuracy.

We decoupled measurement grouping (Chap. 4) and velocity estimation (Chap. 3) in order to focus on each component independently. Then in Chap. 5, we explained our real-time implementation of the overall architecture on cuda-enabled GPUs. All the components necessary for robust scene reconstruction are assembled in this implementation to create a unified module. A multi-object tracking and reconstruction method is prototyped in Chap. 3 which combines raw measurements from LiDAR and camera. We introduced a piece-wise planar surface which serves as an intermediate representation between two sensor spaces. We showed that image-based motion observations (in pixels per second) that exist in image plane are easily translated to observations of three-dimensional velocity (in meters per second) if pixel depth is available. It was also shown that each pixel and each point provide, respectively, 1 and 3 observations on velocity. The proposed Iterative Re-weighted Least Squares (IRLS) finds robust weights for each measurement and linearly combines them. In a sense, the information in all measurements available is combined to perform velocity estimation. The quantitative results prove the robustness of the proposed method when used with different laserscanners. It was tested on datasets collected by 4-layer Scala B2 and 64-layer Velodyne HDL-64E for velocity estimation with/without tracking. An efficient surface-based over-segmentation approach was introduced in Chap. 4, combining accurate but sparse range measurements from a LiDAR with dense image data. As shown in the results, compared to image-based superpixels our supersurfaces are more efficient when geometric information are available for the scene.

Our CUDA implementation made advantage of inherent hierarchical structure and potential for data-level parallelism in the proposed method. We limited the possible set of supersurfaces to overlapping tiles distributed on multi-resolution fusion planes. It was explained how this modification along with others make possible to leverage the full potential of GPUs in this case. For example, we replaced parametric surface reconstruction of Chap. 4 with an edge-preserving depth-map extrapolation. Or, a diffusion-based filtering approach was used instead of α-expansion for multi-label MRF to find tile masks. With a naive set of heuristics, the connectivity of nearby tiles (as supersurfaces) are also analyzed in the GPU implementation – leading to spatially consistent, coherent, and more robust velocity estimates.

There are a few possible directions to improve the current work. For instance, one can couple supersurfaces with a tracking module and refine them through accumulating temporal data. This is similar to the per-pixel Kalman filter method of 6D-Vision [31], but one tracker per tile is enough for our algorithm. As mentioned in Sec. 4.3.2, semantic information could be incorporated for further improvement of segments. Thus instead of the MRF model of Chap. 4 or filtering approach of Chap. 5, tile masks can be inferred from neural networks trained on joint data. Tile connectivity analysis or tile merging can also be investigated in more depth. While a set of heuristics were enough to model tile connections, such parameters could be tuned according to the underlying data. In the simplest case, they can be tuned based on a semantic representation of data. For instance, the in-class horizontal connectivity for pedestrians is generally smaller than vehicles or trucks. This could be inferred from in-place semantic segmentation modules or even trained specific for this task. Although intensity matching is a reliable measurement-level of inferring motion, deep representation are more

robust in cases where brightness constancy assumption is violated. This calls for investigation into inferring 3d velocity observations from high-level deeply-trained representations. Methods like EpicFlow [92] or FlowNet [53] compute 2d optical flow based on deep models, but embedding them into Bayesian framework of this dissertation is both challenging and interesting.

It is possible to make improvements on the hardware side by adding more cameras [116, 108] or even a RADAR. Adding the second camera with overlapping field of view provides disparity measurements, which are useful especially if our choice of LiDAR has low resolution. Unlike other chapters of this thesis, ego-motion is compensated in our CUDA implementation using IMU measurements. This adds a binary classification problem to infer whether each tile/supersurface is stationary or dynamic. This changes the nature of problem into a joint detection-estimation. It is possible to extend this by adding a RADAR to add a multi-hypothesis velocity test to our detection problem.

Bibliography

[1] Autonomous Vehicle Disengagement Report, Department of Motor Vehicles, California. https://www.dmv.ca.gov/portal/dmv/detail/vr/autonomous/disengagement_report_2017.

[2] Ford and Baidu Invest $150 Million in Velodyne for Affordable Lidar for Self-Driving Cars, IEEE Spectrum.

[3] History of Autonomous Cars. https://en.wikipedia.org/wiki/History_of_autonomous_cars.

[4] Lidar Just Got Way BetterâĂŤBut ItâĂŹs Still Too Expensive for Your Car, Technology Review.

[5] Quanergy Announces $250 Solid-State LIDAR for Cars. http://spectrum.ieee.org/cars-that-think/transportation/sensors/quanergy-solid-state-lidar.

[6] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

[7] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. From contours to regions: An empirical evaluation. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2294–2301. IEEE, 2009.

[8] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):898–916, 2011.

[9] 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.

[10] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992.

[11] András Bódis-Szomorú, Hayko Riemenschneider, and Luc Van Gool. Superpixel meshes for fast edge-preserving surface reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2011–2020, 2015.

[12] Jean-Yves Bouguet. Camera Calibration Toolbox for MATLAB. `http://www.vision.caltech.edu/bouguetj/calib_doc/`.

[13] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001.

[14] Thomas Brox, Christoph Bregler, and Jitendra Malik. Large displacement optical flow. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 41–48. IEEE, 2009.

[15] Fatih Calakli and Gabriel Taubin. Ssd: Smooth signed distance surface reconstruction. In *Computer Graphics Forum*, volume 30, pages 1993–2002. Wiley Online Library, 2011.

[16] Mark Campbell, Magnus Egerstedt, Jonathan P How, and Richard M Murray. Autonomous driving in urban environments: approaches, lessons and challenges. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 368(1928):4649–4672, 2010.

[17] Andrea Censi. An icp variant using a point-to-line metric. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 19–25. IEEE, 2008.

[18] Wen-Yan Chang, Chu-Song Chen, and Yi-Ping Hung. Tracking by parts: A bayesian approach with component collaboration. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(2):375–388, 2009.

[19] R Omar Chavez-Garcia, Trung-Dung Vu, and Olivier Aycard. Fusion at detection level for frontal object perception. In *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, pages 1225–1230. IEEE, 2014.

[20] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.

[21] M. H. Daraei, A. Vu, and R. Manduchi. Region segmentation using lidar and camera. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, 2017.

[22] M. Hossein Daraei, Ahn Vu, and Roberto Manduchi. Velocity and shape from tightly-coupled lidar and camera. In *2017 IEEE Intelligent Vehicles Symposium*, 06/2017 2017.

[23] Mohammad Hossein Daraei. Optical flow computation in the presence of spatially-varying motion blur. In *International Symposium on Visual Computing*, pages 140–150. Springer, 2014.

[24] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.

[25] Raúl Dominguez, Enrique Onieva, Javier Alonso, Jorge Villagra, and Carlos Gonzalez. Lidar based perception solution for autonomous vehicles. In *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pages 790–795. IEEE, 2011.

[26] Friedrich Erbs, Alexander Barth, and Uwe Franke. Moving vehicle detection by optimal segmentation of the dynamic stixel world. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 951–956. IEEE, 2011.

[27] Virginia Estellers, Michael Scott, Kevin Tew, and Stefano Soatto. Robust poisson surface reconstruction. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 525–537. Springer, 2015.

[28] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010.

[29] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

[30] Andrew W Fitzgibbon. Robust registration of 2d and 3d point sets. *Image and Vision Computing*, 21(13):1145–1153, 2003.

[31] Uwe Franke, Clemens Rabe, Hernán Badino, and Stefan Gehrig. 6d-vision: Fusion of stereo and motion for robust environment perception. In *Pattern Recognition*, pages 216–223. Springer, 2005.

[32] Brian Fulkerson, Andrea Vedaldi, and Stefano Soatto. Class segmentation and object localization with superpixel neighborhoods. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 670–677. IEEE, 2009.

[33] F. Garcia, A. de la Escalera, and J.M. Armingol. Enhanced obstacle detection based on data fusion for adas applications. In *Intelligent Transportation Systems - (ITSC), 2013 16th International IEEE Conference on*, pages 1370–1375, Oct 2013.

[34] F. Garcia, B. Musleh, A. de la Escalera, and J.M. Armingol. Fusion procedure for pedestrian detection based on laser scanner and computer vision. In *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*, pages 1325–1330, Oct 2011.

[35] Andreas Geiger, Martin Lauer, Christian Wojek, Christoph Stiller, and Raquel Urtasun. 3d traffic scene understanding from movable platforms. *Pattern Analysis and Machine Intelligence (PAMI)*, 2014.

[36] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[37] Andreas Geiger, Frank Moosmann, Ömer Car, and Bernhard Schuster. Automatic camera and range sensor calibration using a single shot. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3936–3943. IEEE, 2012.

[38] Ross Girshick. Fast r-cnn. *arXiv preprint arXiv:1504.08083*, 2015.

[39] 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.

[40] Dale M Grimes and Trevor Owen Jones. Automotive radar: A brief review. *Proceedings of the IEEE*, 62(6):804–822, 1974.

[41] D. Gruyer, A. Cord, and R. Belaroussi. Vehicle detection and tracking by collaborative fusion between laser scanner and camera. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 5207–5214, Nov 2013.

[42] Fatma Guney and Andreas Geiger. Displets: Resolving stereo ambiguities using object knowledge. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4165–4175, 2015.

[43] Fatma Guney and Andreas Geiger. Displets: Resolving stereo ambiguities using object knowledge. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4165–4175, 2015.

[44] Simon Hadfield and Richard Bowden. Kinecting the dots: Particle based scene flow from depth sensors. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2290–2295. IEEE, 2011.

[45] David Held, Devin Guillory, Brice Rebsamen, Sebastian Thrun, and Silvio Savarese. A probabilistic framework for real-time 3d segmentation using spatial, temporal, and semantic cues. In *Proceedings of Robotics: Science and Systems*, 2016.

[46] David Held, Jesse Levinson, Sebastian Thrun, and Silvio Savarese. Combining 3d shape, color, and motion for robust anytime tracking. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

[47] Steven Holmes, Georg Klein, and David W Murray. A square root unscented kalman filter for visual monoslam. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3710–3716. IEEE, 2008.

[48] Lili Huang and Matthew Barth. A novel multi-planar lidar and computer vision calibration procedure using 2d patterns for automated navigation. In *Intelligent Vehicles Symposium, 2009 IEEE*, pages 117–122. IEEE, 2009.

[49] Peter J Huber. *Robust statistics*. Springer, 2011.

[50] Ivan Huerta, Gonzalo Ferrer, Fernando Herrero, Andrea Prati, and Alberto Sanfeliu. Multimodal feedback fusion of laser, image and temporal information. In *Proceedings of the International Conference on Distributed Smart Cameras*, page 25. ACM, 2014.

[51] Jae Pil Hwang, Seung Eun Cho, Kyung Jin Ryu, Seungkeun Park, and Euntai Kim. Multi-classifier based lidar and camera fusion. In *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE*, pages 467–472, Sept 2007.

[52] Eddy Ilg, Rainer Kuummerle, Wolfram Burgard, and Thomas Brox. Reconstruction of rigid body models from motion distorted laser range data using optical flow. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 4627–4632. IEEE, 2014.

[53] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2017.

[54] Bing Jian and Baba C Vemuri. A robust algorithm for point set registration using mixture of gaussians. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1246–1251. IEEE, 2005.

[55] Nico Kaempchen, Matthias Buehler, and Klaus Dietmayer. Feature-level fusion for free-form object tracking using laserscanner and video. In *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*, pages 453–458. IEEE, 2005.

[56] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(7):1409–1422, 2012.

[57] Christian Kerl, Jurgen Sturm, and Daniel Cremers. Dense visual slam for rgb-d cameras. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2100–2106. IEEE, 2013.

[58] Christian Kerl, Jurgen Sturm, and Daniel Cremers. Robust odometry estimation for rgb-d cameras. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3748–3754. IEEE, 2013.

[59] Ronny Klowsky and Michael Goesele. Wavelet-based surface reconstruction from multi-scale sample points. 2014.

[60] Philip Lenz, Julius Ziegler, Andreas Geiger, and Martin Roser. Sparse scene flow segmentation for moving object detection in urban environments. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 926–932. IEEE, 2011.

[61] Alex Levinshtein, Adrian Stere, Kiriakos N Kutulakos, David J Fleet, Sven J Dickinson, and Kaleem Siddiqi. Turbopixels: Fast superpixels using geometric flows. *IEEE transactions on pattern analysis and machine intelligence*, 31(12):2290–2297, 2009.

[62] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 163–168. IEEE, 2011.

[63] Zhengqin Li and Jiansheng Chen. Superpixel segmentation using linear spectral clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1356–1363, 2015.

[64] Ming-Yu Liu, Oncel Tuzel, Srikumar Ramalingam, and Rama Chellappa. Entropy rate superpixel segmentation. In *Computer Vision and Pattern*

Recognition (CVPR), 2011 IEEE Conference on, pages 2097–2104. IEEE, 2011.

[65] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[66] E. Mason, B. Yonel, and B. Yazici. Deep learning for radar. In *2017 IEEE Radar Conference (RadarConf)*, pages 1703–1708, May 2017.

[67] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015.

[68] Pavlos Mavridis, Anthousis Andreadis, and Georgios Papaioannou. Efficient sparse icp. *Computer Aided Geometric Design*, 35:16–26, 2015.

[69] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3061–3070, 2015.

[70] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, 25(9):569–597, 2008.

[71] Alastair P Moore, Simon JD Prince, and Jonathan Warrell. âĂIJlattice cutâĂİ-constructing superpixels using layer constraints. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2117–2124. IEEE, 2010.

[72] Andriy Myronenko and Xubo Song. Point set registration: Coherent point drift. *IEEE transactions on pattern analysis and machine intelligence*, 32(12):2262–2275, 2010.

[73] Fawzi Nashashibi, Ayoub Khammari, and Claude Laurgeau. Vehicle recognition and tracking using a generic multisensor and multialgorithm fusion approach. *International Journal of Vehicle Autonomous Systems*, 6(1):134–154, 2008.

[74] Dominik Nuss, Markus Thom, Andreas Danzer, and Klaus Dietmayer. Fusion of laser and monocular camera data in object grid maps for vehicle environment perception. In *Information Fusion (FUSION), 2014 17th International Conference on*, pages 1–8. IEEE, 2014.

[75] Peter Ondruska and Ingmar Posner. Deep tracking: Seeing beyond seeing using recurrent neural networks. *arXiv preprint arXiv:1602.00991*, 2016.

[76] Jeremie Papon, Alexey Abramov, Markus Schoeler, and Florentin Worgotter. Voxel cloud connectivity segmen/tation-supervoxels for point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2027–2034, 2013.

[77] Bojan Pepik, Peter Gehler, Michael Stark, and Bernt Schiele. 3d2pm–3d deformable part models. In *Computer Vision–ECCV 2012*, pages 356–370. Springer, 2012.

[78] Anna Petrovskaya and Sebastian Thrun. Model based vehicle tracking for autonomous driving in urban environments. *Proceedings of Robotics: Science and Systems IV, Zurich, Switzerland*, 34, 2008.

[79] C. Premebida, G. Monteiro, U. Nunes, and P. Peixoto. A lidar and vision-based approach for pedestrian and vehicle detection and tracking. In *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE*, pages 1044–1049, Sept 2007.

[80] Victor A Prisacariu and Ian D Reid. Pwp3d: Real-time segmentation and tracking of 3d objects. *International journal of computer vision*, 98(3):335–354, 2012.

[81] Victor Adrian Prisacariu, Aleksandr V Segal, and Ian Reid. Simultaneous monocular 2d segmentation, 3d pose recovery and 3d reconstruction. In *Computer Vision–ACCV 2012*, pages 593–606. Springer, 2013.

[82] Siyang Qin, Seongdo Kim, and Roberto Manduchi. Automatic skin and hair masking using fully convolutional networks. In *Multimedia and Expo (ICME), 2017 IEEE International Conference on*, pages 103–108. IEEE, 2017.

[83] Siyang Qin and Roberto Manduchi. A fast and robust text spotter. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*, pages 1–8. IEEE, 2016.

[84] Siyang Qin and Roberto Manduchi. Cascaded segmentation-detection networks for word-level text spotting. *arXiv preprint arXiv:1704.00834*, 2017.

[85] Siyang Qin, Peng Ren, Seongdo Kim, and Roberto Manduchi. Robust and accurate text stroke segmentation. In *Applications of Computer Vision (WACV), 2018 IEEE Winter Conference on*, pages 242–250. IEEE, 2018.

[86] Julian Quiroga, Thomas Brox, Frédéric Devernay, and James Crowley. Dense semi-rigid scene flow estimation from rgbd images. In *Computer Vision–ECCV 2014*, pages 567–582. Springer, 2014.

[87] Julian Quiroga, Frédéric Devernay, and James Crowley. Local scene flow by tracking in intensity and depth. *Journal of Visual Communication and Image Representation*, 25(1):98–107, 2014.

[88] Clemens Rabe, Uwe Franke, and Stefan Gehrig. Fast detection of moving objects in complex scenarios. In *Intelligent Vehicles Symposium, 2007 IEEE*, pages 398–403. IEEE, 2007.

[89] Clemens Rabe, Thomas Müller, Andreas Wedel, and Uwe Franke. Dense, robust, and accurate motion field estimation from stereo image sequences in real-time. In *Computer Vision–ECCV 2010*, pages 582–595. Springer, 2010.

[90] Kaushik Raghu, Mohammadhossein Daraei, Premkumar Natarajan, and Norman Lopez. Early detection of exit only and shared lanes using perception technology, June 23 2016. US Patent App. 14/579,144.

[91] 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.

[92] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow. In *Computer Vision and Pattern Recognition*, 2015.

[93] Terry Scott, Akshay A Morye, Pedro Piniés, Lina M Paz, Ingmar Posner, and Paul Newman. Exploiting known unknowns: Scene induced cross-calibration of lidar-stereo systems. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 3647–3653. IEEE, 2015.

[94] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics: Science and Systems*, volume 2, 2009.

[95] Mark Sheehan, Alastair Harrison, and Paul Newman. Self-calibration for a 3d laser. *The International Journal of Robotics Research*, page 0278364911429475, 2011.

[96] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[97] Ali Siahkoohi, Rajiv Kumar, and Felix J. Herrmann. Seismic data reconstruction with generative adversarial networks. In *80th EAGE Conference and Exhibition 2018*, 2018.

[98] Luciano Spinello, Rudolph Triebel, and Roland Siegwart. Multiclass multimodal detection and tracking in urban environments. *The International Journal of Robotics Research*, 29(12):1498–1515, 2010.

[99] Deqing Sun, Erik B Sudderth, and Hanspeter Pfister. Layered rgbd scene flow estimation. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 548–556. IEEE, 2015.

[100] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998.

[101] Mohammad Hossein Daraei Haji Tooei. *Investigation into optical flow problem in the presence of spatially-varying motion blur*. University of California, Santa Cruz, 2014.

[102] Yanghai Tsin and Takeo Kanade. A correlation-based approach to robust point set registration. In *European conference on computer vision*, pages 558–569. Springer, 2004.

[103] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[104] Michael Van den Bergh, Xavier Boix, Gemma Roig, Benjamin de Capitani, and Luc Van Gool. Seeds: Superpixels extracted via energy-driven sampling. In *European conference on computer vision*, pages 13–26. Springer, 2012.

[105] A. Vatavu, R. Danescu, and S. Nedevschi. Stereovision-based multiple object tracking in traffic scenarios using free-form obstacle delimiters and particle filters. *Intelligent Transportation Systems, IEEE Transactions on*, 16(1):498–511, Feb 2015.

[106] Sundar Vedula, Simon Baker, Peter Rander, Robert Collins, and Takeo Kanade. Three-dimensional scene flow. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 722–729. IEEE, 1999.

[107] Olga Veksler, Yuri Boykov, and Paria Mehrani. Superpixels and supervoxels in an energy optimization framework. *Computer Vision–ECCV 2010*, pages 211–224, 2010.

[108] Christoph Vogel, Konrad Schindler, and Stefan Roth. Piecewise rigid scene flow. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1377–1384, 2013.

[109] Trung-Dung Vu, O. Aycard, and F. Tango. Object perception for intelligent vehicle applications: A multi-sensor fusion approach. In *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, pages 774–780, June 2014.

[110] Dominic Zeng Wang, Ingmar Posner, and Paul Newman. Model-Free Detection and Tracking of Dynamic Objects with 2D Lidar. *The International Journal of Robotics Research (IJRR)*, 2015.

[111] Andreas Wedel, Clemens Rabe, Tobi Vaudrey, Thomas Brox, Uwe Franke, and Daniel Cremers. *Efficient dense scene flow from sparse or dense stereo data*. Springer, 2008.

[112] H. Weigel, P. Lindner, and G. Wanielik. Vehicle tracking with lane assignment by camera and lidar sensor fusion. In *Intelligent Vehicles Symposium, 2009 IEEE*, pages 513–520, June 2009.

[113] David Weikersdorfer, David Gossow, and Michael Beetz. Depth-adaptive superpixels. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 2087–2090. IEEE, 2012.

[114] Kevin Wyffels and Mark Campbell. Joint tracking and non-parametric shape estimation of arbitrary extended objects. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3360–3367. IEEE, 2015.

[115] Yu Xiang, Changkyu Song, Roozbeh Mottaghi, and Silvio Savarese. Monocular multiview object tracking with 3d aspect parts. In *Computer Vision–ECCV 2014*, pages 220–235. Springer, 2014.

[116] Koichiro Yamaguchi, David McAllester, and Raquel Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. In *Computer Vision–ECCV 2014*, pages 756–771. Springer, 2014.

[117] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Pattern Recognition*, pages 214–223. Springer, 2007.

[118] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2174–2181. IEEE, 2015.

[119] Qilong Zhang and Robert Pless. Extrinsic calibration of a camera and laser range finder (improves camera calibration). In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2301–2306. IEEE, 2004.

[120] Qilong Zhang and Robert Pless. Extrinsic calibration of a camera and laser range finder (improves camera calibration). In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2301–2306. IEEE, 2004.

[121] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015.

[122] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *arXiv preprint arXiv:1711.06396*, 2017.