Semantic Mapping for Autonomous Navigation and Exploration

Daniel Maturana

CMU-RI-TR-21-55

August 5th, 2021

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Sebastian Scherer, Ph.D, Chair
Martial Hebert, Ph.D
Abhinav Gupta, Ph.D
Raquel Urtasun, Ph.D, University of Toronto and Waabi

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

The author gratefully acknowledges the support from the United Technologies Research Center, the Office of Naval Research, the Yamaha Motor Corporation, and the Qualcomm Innovation Fellowship.

Abstract

The last two decades have seen enormous progress in the sensors and algorithms for 3D perception, giving robots the means to build accurate spatial maps and localize themselves in them in real time. The geometric information in these maps is invaluable for navigation while avoiding obstacles, but insufficient, by itself, for robots to robustly perform tasks according to human goals and preferences. Semantic mapping is a promising framework to provide robots with a richer representation of their environment by augmenting spatial maps with semantic labels – in other words, a map of what is where. However, for semantic maps to fulfill their potential to improve robotic capabilities, we need systems capable of building and continuously updating these maps from noisy and ambiguous sensor streams with acceptable levels of accuracy and latency. In this thesis, we make several contributions to address these challenges and demonstrate their benefits in real-world scenarios.

First, we introduce a system for real-time semantic mapping from low-altitude aerial lidar that explicitly models the ground surface to extract more robust point cloud features. We show this approach improves the classification accuracy of relevant categories for safe navigation of an autonomous helicopter in human-populated environments.

Second, we advance the state of the art in point cloud classification by moving away from hand-engineered features with VoxNet, a novel deep learning architecture based on 3D Convolutional Neural Networks (CNNs) that learns features and classifiers directly from a volumetric representation. VoxNet outperforms various baselines for the task of mapping safe landing zones for a helicopter in cluttered terrain, and sets the state of the art in 3D object recognition benchmarks from three different domains.

Third, we develop two systems for multimodal semantic mapping with camera imagery and lidar point clouds. The first system implements a fast decoupled strategy, where image and lidar are used to infer semantic labels and elevation maps, respectively. The second system learns to fuse both modalities with a novel joint 2D/3D CNN architecture for semantic segmentation. We apply these systems to the task of off-road navigation with an autonomous all-terrain vehicle, allowing it to traverse cluttered and narrow trails in off-road environments.

Finally, we develop a lightweight semantic mapping system for Micro-Aerial Vehicles (MAVs) with payload constraints that preclude lidar and high-powered computing platforms. We propose a novel 2.5D mapping system that takes advantage of publicly available digital elevation maps and priors of object height to achieve real-time mapping of distant objects using camera imagery. We show this system enables significant time savings in the task of autonomously gathering information for semantic classes of interest with MAVs.

Overall, our work – which spans three robotic platforms, four different tasks, and a wide range of sensing and computing capabilities – shows that semantic mapping is a versatile and pragmatic framework to extend and improve robotic abilities.

Acknowledgments

I would like to thank my advisor, Sebastian Scherer, who many years ago took a chance on me as the first doctoral student in his nascent lab. It has been amazing to see the growth and consolidation of the AirLab over the years, and I am honored to be one of its founding members. Without Sebastian's support, advice, and guidance this thesis would have never come to fruition.

I would also like to thank Abhinav Gupta, Martial Hebert, and Raquel Urtasun for serving on my committee. As superstar researchers with many commitments inside and outside of academia, their time is extremely valuable, and I am deeply grateful that they set aside some of it for me.

Throughout my PhD, I was fortunate to work with many brilliant and hard-working collaborators. Without the help of Sankalp Arora, Geetesh Dubey, Dong-Ki Kim, Po-Wei Chou, Sezal Jain, Yu Song, Uenoyama Masashi, Greg Armstrong and Adam Stambler, many results in this thesis would not have been possible. Outside of my thesis work, I also learned a great deal collaborating with Ratnesh Madaan, Shichao Yang, Kristen Holz, Xiangwei Wang, David Fouhey, and Alyosha Efros, among others.

I must also acknowledge some of the many mentors who helped me get started in my academic journey, without whose help I would have likely not have started a PhD in the first place: my undergraduate advisor, Álvaro Soto, as well as my early collaborators Domingo Mery, Nick Roy, Abe Bachrach and Albert Huang.

I would also like to thank the many friends that enriched my life inside and outside the lab: David Fouhey, Michael Shomin, Sankalp Arora, Geetesh Dubey, Paul Vernaza, Shervin Javdani, JP Mendoza, Cristóbal de Araujo, JF Winkles, Marie Zimmerman, and too many others to list here; you know who you are[1].

Finally, I would like to thank my family and loved ones that were always there for me throughout this process. I especially would like to thank my parents Gina and Sergio, as well as my siblings, who have always been supportive and encouraging in all my endeavors; my wife Kathryn, who has always been endlessly patient and loving; and my cat Babette, who always kept me company during long nocturnal coding sessions.

[1] If you are in doubt, rest assured I consider you a friend.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

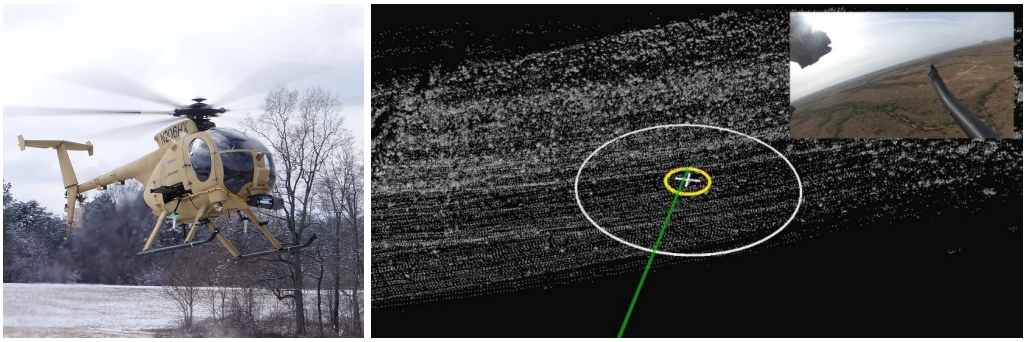
1.1 A critique of pure geometry

Ubiquitous mapping and GPS localization, inexpensive high-quality cameras and depth sensors, and increasingly fast and accurate 3D reconstruction technology have made it feasible for modern robots to have a detailed, real-time sense of the spatial geometry of their surroundings. This information is a key input to most of the planning algorithms that allow robots to find efficient routes and safely avoid obstacles.

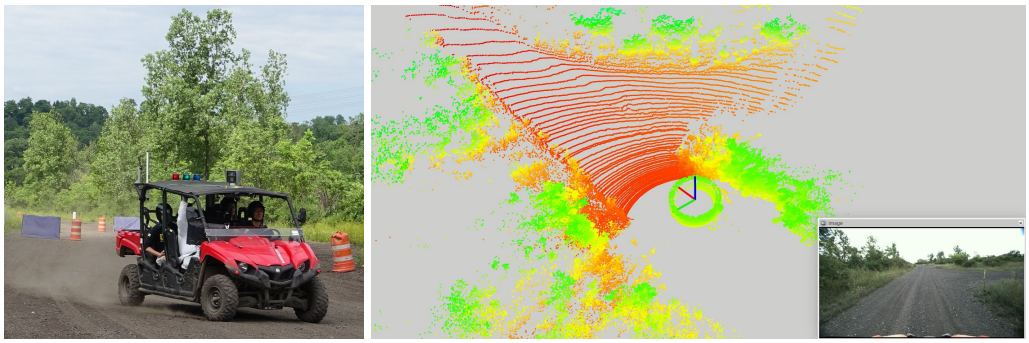
However, the performance of robots based purely on this information is often disappointing. Consider these scenarios, based on the work in this thesis (Figure 1.1):

- A lidar-equipped autonomous helicopter that selects suitable landing zones by finding planar or near-planar surfaces in the point cloud may decide that a building's roof top is a preferable landing zone to a nearby, less planar, location on the ground. A human pilot would usually choose the latter, as most roof tops are not permissible landing zones. Without any knowledge of which surfaces may be *roof* or *ground*, the helicopter is ill-equipped to make the right choice.
- An autonomous all-terrain vehicle (ATV) that encounters a small patch of grass along its intended path stops to halt, as it appears to its depth sensors as an obstacle or extremely rough terrain. A human driver, knowing the grass is soft, would have decided to drive over it, perhaps after slowing down.
- An autonomous Micro Aerial Vehicle (MAV) is tasked with finding and capturing high-resolution imagery of objects belonging to a specific semantic category, such as cars. In this situation, a human pilot with a first-person camera view can take advantage of their knowledge of what cars look like (and where they are likely to be located) to find and fly towards cars to capture close-up images. In contrast, a MAV with no semantic awareness of the environment must resort to an exhaustive aerial survey of the area, at a significantly higher cost in flight time.

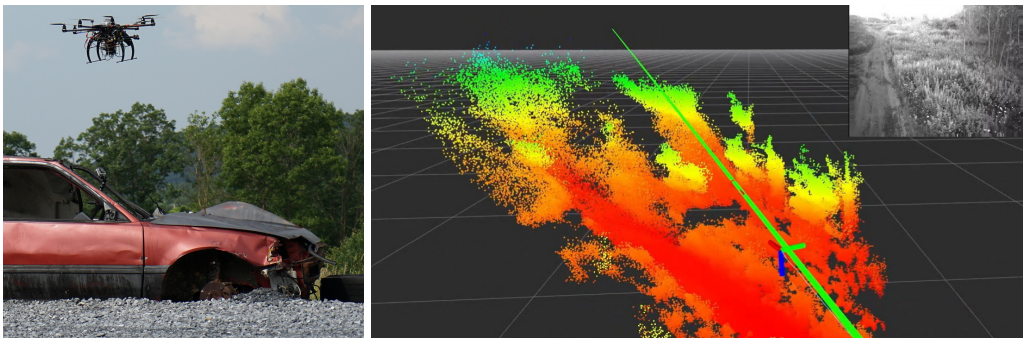
These examples illustrate that representations of the world based purely on spatial information do not provide a sufficiently expressive basis for robots to effectively carry out tasks according to human goals and preferences.



(a) The Unmanned Little Bird autonomous helicopter (chapter 3, chapter 4)



(b) The Erik autonomous All-Terrain Vehicle (chapter 5)



(c) The MAVScout autonomous quadrotor (chapter 6)

Figure 1.1: Robots featured in this thesis and their sensor data.

1.2 Towards a richer inner world: semantic mapping

The biologist von Uëxkull introduced the notion of *Umwelt* for the study of animal behavior, defined roughly as the world as experienced by the animal, and often translated as "self-world" or "subjective inner world" [52]. The animal's Umwelt depends on its sense organs, and more importantly, on its own biological needs; it represents "the set of things in the world that matter to it and which it needs to discriminate and anticipate as best as its can", in the words of the philosopher D. Dennett [46, p. 11]. Each animal will build an Umwelt that carries the specific *meanings* — *i.e.*, *semantics* — about the environment it needs to act in an adaptive way, and the scope of the animal's Umwelt sets the limits of its behavior.

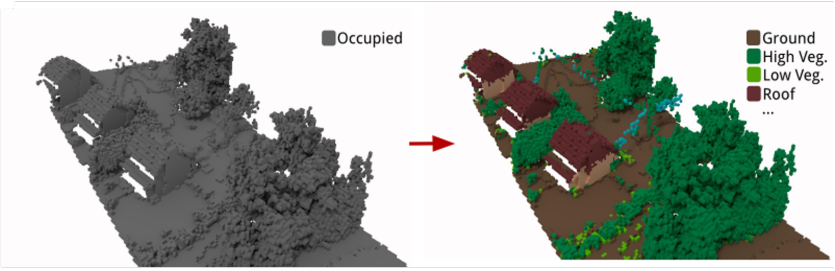


Figure 1.2: Spatial map versus semantic map. In addition to representing spatial geometry of the environment, semantic maps represent other semantic attributes, such as object categories.

Applying this view to robots, what "semantics" should their Umwelt carry? If we assume that for robots, acting adaptively means accomplishing human-specified goals, then semantics that relate to a human's own view of the environment are a natural choice. While a representation of the environment that matches human levels of nuance are still out of reach, the examples above suggest that even a simple step beyond purely spatial maps — spatial maps annotated with category-level semantics — can significantly extend the capabilities of robots. Such a representation is often known as a *semantic map* [65, 133, 96, 178, 104], and the process of creating this map as *semantic mapping*.

In this thesis, we investigate semantic mapping as a framework to extend and improve the capabilities of robots across different tasks and scenarios. In general, the goal of a semantic mapping system is to create a map that represents not only information about the spatial geometry of the environment, but also some type of "meaning" — *i.e.*, semantics — associated with the spatial features. In this work, we will restrict ourselves to semantic maps defined by two design choices: First, we will assume that spatial information will be in the form of three-dimensional metric maps; second, we assume the semantic information will correspond to a fixed set of categories chosen *a priori*, based on the needs of each task. Figure 1.2 shows such a map.

Both of these choices are rooted in the same assumption: that we are designing robots to accomplish tasks that are useful for humans, and thus, we are tailoring their inner world in a way that is transparent for humans. Metric representations, while certainly not the only spatial representation that is interpretable for humans, are — by design — a universal, objective way for humans to conceptualize and communicate about space. In robotics, metric representations are heavily used in mapping, planning, and control, especially in the temporal and spatial scales that concern the robotic tasks we target in this thesis. Likewise, we argue that categories are also an

effective and pragmatic way to represent a semantic interpretation of the world that is shared by the robot and humans. While less objective than spatial units of measurement, humans can usually agree on how to categorize various entities [150]. In addition, using category-level predictions allows us to leverage the large body of machine learning research and computer vision that adopts this paradigm in the form of classification and semantic segmentation. Naturally, these design choices have certain drawbacks, which we discuss further in chapter 7.

Before we summarize the contributions in this thesis, let us return to our example scenarios to illustrate how semantic maps can be used in practice:

- In the helicopter scenario, we create a 3D voxel map that predicts a semantic label for each occupied voxel, where the possible labels include *roof*, *ground*, *tree canopy*, etc. This representation can be readily used by the planner to avoid roofs as a landing zone.
- Similarly, in the ATV scenario, we create a local 3D voxel map where each voxel is labeled as one of *trail*, *low vegetation*, etc.; using these predictions, a planner can create a more natural route through rough terrain.
- In the MAV scenario, we use camera imagery to maintain a global elevation map with probabilistic estimates for the presence and coarse location of cars (or any given class of interest) around the robot. This map is used by the MAV's viewpoint-aware planner to create trajectories that enable the MAV to search for the class of interest and capture high-quality images for instances of the class whenever it is detected.

1.3 Contributions and overview

Implementing semantic mapping systems that significantly extend robotic capabilities presents several challenges. The systems must be able to make predictions with sufficient accuracy for each task in the presence of low interclass variability and high intraclass variability, and be robust to noise and outliers in the sensor data. Moreover, they must make these predictions with low enough latency to provide the planning system with the relevant information to choose the next action.

In this thesis, we propose several contributions to the state of the art in semantic mapping with image and point cloud data. For each contribution, we demonstrate its benefits for real-world robotic applications:

Ground surface-aware semantic mapping (chapter 3). As our first contribution, we develop a real-time semantic mapping system for low-altitude aerial point clouds, which predicts various prominent classes in urban and suburban environments, such as *ground*, *tree*, *building roof*, *building wall*, etc.

Our semantic mapping pipeline builds on the prior state of the art in semantic segmentation for point cloud segmentation, but improves on this baseline in two ways. First, motivated by the importance of the ground surface as a discriminative cue for semantic inference, we propose a specialized 2.5D ground surface estimation method resistant to the significant occlusion and sparsity present in aerial lidar point clouds, and use it to improve semantic classification. Second, we implement an intermediate grid-based representation encoding sufficient statistics for point clouds that enables efficient incremental updates and extraction of features for semantic classification.

Our motivating application is to provide an autonomous full-sized helicopter with semantic awareness of classes relevant for the planning of low-altitude flight trajectories and landing. We evaluate our approach on a custom dataset captured with a lidar-equipped helicopter, and show the benefits of incorporating ground surface estimation in the pipeline. In addition, we demonstrate real-time operation of our pipeline on streaming point clouds.

Deep learning for robust and efficient point cloud classification (chapter 4). Our second contribution aims to overcome a shortcoming of the earlier state-of-the-art in point cloud labeling systems, including our first contribution. These systems often struggle to discriminate between categories that appear highly similar in point clouds, despite using a diverse set of features designed to capture various kind of relevant cues for classification tasks. Under the hypothesis that the hand-engineered point cloud features used by these systems do not encode sufficient task-specific information, we developed *VoxNet*, a novel end-to-end deep learning approach for point cloud classification that integrates volumetric occupancy maps with spatially 3D Convolutional Neural Networks (CNNs). Like other deep learning approaches, this approach can learn how to extract discriminative features at multiple levels of abstraction from the data, enabling higher classification accuracy and easing the burden of feature design.

We evaluate this system in the challenging task of safe landing zone detection for helicopters, showing real-time performance and a significant improvement over systems with hand-engineered features. Furthermore, we show the benefits of our architecture in more generic object recognition tasks by improving over the state of the art in three benchmarks, each using a different source of 3D data (lidar, RGBD sensing, and CAD models).

Multimodal semantic mapping with image and point cloud data (chapter 5). In our third contribution, we investigate the fusion of camera imagery and point cloud data for semantic mapping. While we showed point cloud data from lidar can be used effectively for semantic as well as spatial inference, acquiring point clouds with the density and accuracy required to discriminate certain classes may be infeasible with lower end sensors. On the other hand, image sensors that provide high-resolution color and texture data are widely available at a relatively low cost and footprint. Thus, we are interested in taking advantage of the complementary information provided by these two modalities to improve semantic mapping.

In the first part of the contribution, we investigate how to use point cloud and image data jointly for semantic inference. We propose a novel multimodal CNN architecture that learns to fuse sensor information from image and point cloud data. The network features a 2D CNN stream and a 3D CNN stream, and the two are interconnected with learned projection modules. We apply this architecture to the problem of 2D semantic segmentation for autonomous off-road navigation from image and lidar data, and show its accuracy benefits over various baselines in offline experiments.

In the second part of this contribution, we propose a multimodal semantic mapping system that uses a decoupled approach to fusing the information from point cloud and image data, with the aim of obtaining real-time operation. Based on the cost/accuracy trade-offs we observed in our initial approach, we opt to exploit the relative strengths of each modality, using images for semantic segmentation with a custom 2D CNN and lidar for

spatial mapping with a 2.5D elevation map. We apply the system to the problem of robust path following in off-road environments for an autonomous All-Terrain Vehicle (ATV). We evaluate the performance of our 2D CNN in offline experiments on two datasets, including a custom dataset gathered with our platform, and show comparable accuracy to state-of-the-art approaches at lower latency. We also demonstrate autonomous operation of our system in the field, where the semantic maps enable the ATV to follow a variety of challenging off-road trails.

Long-range semantic mapping for semantic exploration (chapter 6). Our prior contributions in this thesis rely on range sensors to acquire the spatial information for our semantic maps. However, a drawback of this approach is that current range sensors quickly become sparse, inaccurate, or both at longer distance ranges.

In our final contribution, we investigate how to create long-range semantic maps in real time for based on high-resolution camera imagery and global positioning estimates. We are motivated by a novel task, *semantic exploration*, where the goal is to enable the robot to efficiently acquire high-resolution sensor data for a given class of interest within a large area. This requires the robot to locate the objects of interest with onboard sensing before approaching them for high-resolution data acquisition. This task is relevant to applications such as inspection, search and rescue, surveillance, etc.

Our hypothesis is that we can use a forward-facing camera to estimate the presence and location of potentially distant instances of the class of interest, and use these estimates to guide the robot towards these locations to gather high-quality data. Due to their agility and ability to easily capture camera imagery from different viewpoints and altitudes, Micro-Aerial Vehicles (MAVs) are an attractive platform to explore this approach. However, their relatively low payload capacity restricts us to relatively limited computing platforms and precludes the use of lidar.

To enable effective semantic exploration within these constraints, we propose a system combining a lightweight 2D segmentation CNN with a coarse 2.5D height mapping system that uses global positioning estimates and preexisting Digital Elevation Maps (DEMs) to predict the location of objects of interest. We evaluate the performance of the semantic segmentation and mapping modules on two custom datasets, and demonstrate successful semantically exploration in the field.

Together, the contributions in this thesis – which spans three robotic platforms, four different tasks, and a wide range of sensing and computing capabilities – show that semantic mapping is a versatile and pragmatic framework that allows us to extend and improve robotic abilities by providing them with a rich human-compatible representation of their environment in real time.

Chapter 2

Background

The goal of the semantic mapping systems in this thesis is to provide robots with a coherent, real-time representation of their environment encoding the spatial and semantic structure they need for effective action. To meet this goal, these systems must fuse information from multiple sources into a global metric representation that is suitable for use with the robot's navigation planning systems, online and in real time. To handle the complexity of this task, we adopt a modular approach, with separate subsystems for spatial mapping and semantic inference, and the fusion of the respective outputs of these systems into a semantic map. This allows us to adopt and build on the prior state of the art for each of these tasks, whenever possible.

In this chapter, we will introduce the tools that we build on for these subsystems and how we use them in our semantic mapping pipeline. Section 2.1 briefly surveys the technologies and algorithms in spatial perception that we rely on for our semantic maps. Section 2.2 discusses the most common tasks in computer vision and machine learning that relate to semantic mapping, and summarizes relevant prior work in these areas.

2.1 Tools for spatial perception

As we discussed in chapter 1, the technology and algorithms for spatial perception have made enormous strides in the past two decades, and currently provide several mature tools that simplify the task of estimating the robot's pose and acquiring spatial maps of the environment in real time. As a consequence, for the most part we will rely on existing solutions for spatial localization and mapping, and focus on improving the inference of semantic properties. However, given their importance to the work in this thesis, we will briefly introduce the tools we utilize and how we use their outputs for semantic mapping.

2.1.1 The spatial information we need

For the work in this thesis we assume, unless otherwise stated, that we have access to the following (somewhat idealized) pieces of knowledge regarding spatial state:

- A real-time stream of 6-degree of freedom[1] pose measurements of the robot, time-stamped and aligned to a geocentric frame[2]
- A real-time stream of 3D point clouds, temporally synchronized with the pose stream, and aligned to the same geocentric frame

These are relatively specific requirements, and in practice can be partially relaxed for some of our applications. For example, with the exception of the system in chapter 6, we do not explicitly depend on the geocentric alignment of our spatial frame of reference; any fixed, gravity-aligned frame is sufficient. Likewise, we don't necessarily depend on the point clouds being aligned to the same frame as the pose, as long as we have the necessary information to transform the point clouds (or spatial data that can be converted into point clouds, such as depth maps) to be in a globally consistent frame.

We have not made any explicit requirements on the accuracy or precision of these measurements. In practice, throughout these thesis we simply use the highest-quality spatial information available to us in the various robotic platforms we use, and engineer our semantic pipeline as needed in each case. The quality of the spatial measurements is primarily constrained by the sensors and computational capacity that each platform has at its disposal, and there is a wide variation in quality, as we discuss below.

While stating these requirements is simple, fulfilling them is far from trivial. In the next sections we survey the hardware and software technologies that have made it possible to obtain this spatial information our semantic mapping systems depend on.

2.1.2 3D sensing

As part of a robot's perception system, the design of any semantic mapping system will naturally be deeply influenced by the robot's sensors, as they provide the starting point for both spatial and semantic perception of the environment. Nowadays, robot engineers have a vast array of sensors at their disposal, allowing robots to perceive a wide variety of properties of the environment. Despite their differences in size, function, and operating environment, the robots that we encounter in this thesis use many of the same sensing technologies. In particular, they use different types of inertial sensors, global positioning sensors, and range sensors; together, these sensors enable the robots to build metric 3D representations of the world in real time, the availability of which is an assumption that we make throughout this thesis.

Here we briefly introduce these sensors, their different trade-offs, and how they impact the semantic mapping in this thesis. For a comprehensive textbook on INS and GNSS, see Groves [72]. Kelly [89, chap. 6 and 8] provides a useful introduction to inertial sensing systems and range sensing for robotics. Szeliski [168, chap. 11] for an introduction to stereo vision, and Zollhöfer [193] for an introductory survey of RGB-D sensors.

[1] Three translational degrees of freedom and three rotational degrees of freedom; for example, $(x, y, z, roll, pitch, yaw)$.

[2] An earth-centric coordinate frame; in practice, this means a frame that we can use as an "absolute" and unambiguous frame of reference on the planet where our research takes place.

Inertial Sensors and global positioning

Broadly speaking, inertial sensors allow the robot to perceive its own pose and/or motion relative to an inertial frame of reference. The most well-known sensors of these type are accelerometers and gyroscopes. Accelerometers measure external forces acting on the vehicle (including gravity), which can be used to estimate linear acceleration. Gyroscopes measure angular velocity relative to an inertial frame of reference. Most modern robots will use Micro-Electro-Mechanical Systems (MEMS) versions of these sensors, integrated into a single package known as Inertial Measurement Units (IMUs). Most IMUs used in robotics will have three accelerometers and three gyroscopes, arranged to provide measurements for six degrees of freedom (three of linear motion and three of rotational motion).

By integrating the gyroscopic and accelerometer measurements over time, it is possible to estimate the robot's attitude (pitch and roll relative to gravity) as well as its 6-dof pose relative to an arbitrary fixed frame. Systems that perform this estimate with IMUs are often known as Inertial Navigation Systems or INS. While the term is often used to describe integrated packages that include sensors and embedded processors for state estimation, it can be applied more broadly to systems that perform the same functionality with independent sensing and processing hardware.

INS typically provide high-rate ($>50\,\mathrm{Hz}$) pose estimates that are critical for robotic control, and an important cue towards estimating the robot's pose in a global frame. However, this estimate will drift over time, as small errors in integration of the rotational and linear motions, plus other sources of noise, compound. The rate at which this happens will depend on the IMU performance, for which there is a wide range, but for most practical purposes the pose estimates of an IMU will be useless for 3D mapping by themselves.

To mitigate this problem, many IMUs include additional sensors. A common additional sensor is a 3-axis magnetometer (compass), which provides an absolute reference for orientation in the yaw axis. Another is a barometer, which can be used to estimate altitude, specially useful for aerial robots. IMUs with these sensors are sometimes referred to as 9-dof and 10-dof IMUs, respectively. For wheeled vehicles, wheel encoders are often used to estimate motion. When used for INS, these sensors are sometimes referred to as aiding sensors.

Even these additional sensors are not a solution to the issue of drift; not only are they often noisy, but still do not provide an absolute reference for 3D position. The most common solution to this problem, at least in outdoor scenarios, are Global Satellite Navigation Systems (GNSS), of which the most well known is the Global Positioning System (GPS). GNSS are based on the computation of time delays between radio signals from a set of satellites orbiting Earth; when sufficient signals are received, the vehicle can fix its position on the Earth surface to within $5\,\mathrm{m}$ to $20\,\mathrm{m}$, depending on the quality and quantity of the signals.

The combination of GPS and IMU is extremely useful for robots operating in outdoor environments, as it allows the robot to estimate its 6-dof pose in a global geocentric frame using compact, lightweight hardware and with a computational cost that is negligible by modern standards. However, depending on the application and the INS performance, the accuracy obtained by this solution is still insufficient. To improve the accuracy of the estimated position at the cost of additional infrastructure, Differential GPS (DGPS) and Real-Time-Kinematic (RTK) systems use additional radio communication with ground stations with known positions. These systems can achieve centimeter-level accuracy.

However, in practice, even systems with this level of accuracy usually need to supplement their positioning systems with other sources of information. Even if the robot knows its position with perfect accuracy, the position of the objects (and potential obstacles) around it are likely not known with the same level of accuracy, or in many cases not known at all (*e.g.*, a self-driving car may have a road map in GPS coordinates, but will probably not know the GPS coordinates of pedestrians or other cars). Moreover, in the real world, these systems are often disrupted in various ways. As examples, in urban environments or GPS radio signals may be blocked or affected by multipath interference; accelerometer measurements may be highly inaccurate under high-frequency vibration; compass measurements may be disturbed by the presence of large amounts of metal in the environment [3]. As we discuss in subsection 2.1.3, INS is usually supplemented with mapping and localization algorithms based on other sensor modalities.

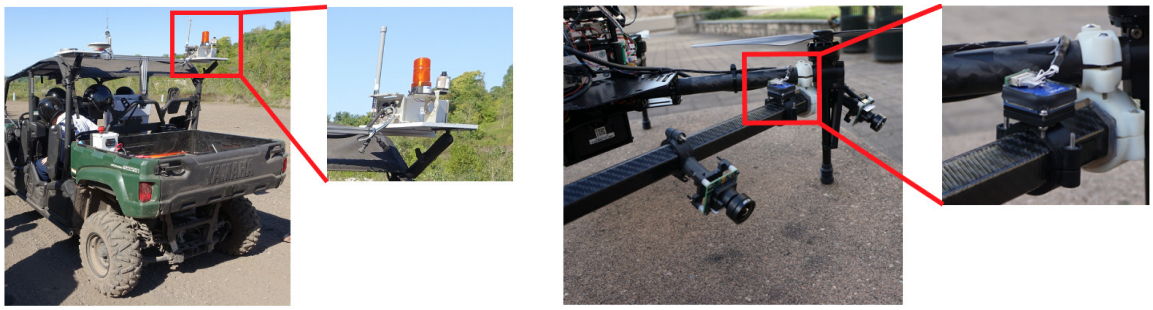


Figure 2.1: Left: integrated INS used in the ATV platform of of chapter 5. Right: the IMU mounted on the Micro Aerial Vehicle platform of chapter 6.

All the robotic systems featured in this thesis use INS with these sensors, albeit with a wide range of performance characteristics, ranging from the aerospace-grade INS in the helicopter of chapter 3 to the compact consumer-grade sensors of the Micro-Aerial Vehicle in chapter 6, with the automotive-grade INS of the All-Terrain Vehicle from chapter 5 falling in between these extremes. Figure 2.1 depicts the hardware components of the INS systems used in chapter 5 and chapter 6, minus the antennas that improve their reception of GNSS signals.

Range sensing

The last section discussed sensors that help the robot estimate its own position, and thus, at least in theory, can cover the first requirement we specified in the beginning of this chapter. But building spatial maps also requires perceiving the position and shape of surfaces in its surroundings.

Various types of sensors have emerged to achieve this purpose; collectively, they are often referred to as range sensors or depth sensors, as they measure distance or range from the sensor to each surface. The measurements of range and depth sensors are often aggregated in the form of point clouds; thus, in this thesis we often use the term "point cloud data" as a shorthand for the kind of data that is created by these different sensors.

[3] Interference from metals may come from unexpected places. We learned this lesson first-hand, when we saw the iron-rich soil in one of our testing locations in Pittsburgh — known as the "steel city" for a reason — wreak havoc with our magnetometers .

The range sensor that will feature most prominently in our work is lidar, but we will also encounter structured light sensors and stereo sensors. In recent years, these sensors have seen dramatic improvements in sensing quality, as well as size and cost, making them extremely useful for robotic applications. Many results in this thesis are enabled by the availability of high-quality 3D range sensing, which simplifies the construction of detailed 3D maps of the environment.

However, while much work has been done in taking advantage of these sensors for spatial mapping tasks (subsection 2.1.3), there is still relatively little research on semantic perception tasks with these sensors, which is one of the gaps this thesis aims to address. We briefly introduce these sensors below.

Lidar Lidar – an abbreviation of LIght Detection And Ranging[4] – has seen a massive increase in adoption over the last few years, largely fueled by its applications in autonomous self-driving.

Lidar is an active range sensor that works with a time-of-flight principle on emitted light (laser) signals. The laser signal is modulated (usually as pulses) and the distance from the sensor to the surface it is pointing at is estimated by measuring the time the modulated signal is reflected back to the sensor.

Most lidar devices are "single-pixel" devices, *i.e.*, they measure range for a single direction at any given time. However, the beam width of this measurement (also known as the instantaneous field of view) is typically very narrow, and the time it takes to perform a single measurement is very short — for example, the lidar sensor used in the helicopter platform from chapter 3 has a beam width of $0.35\,\mathrm{mrad}$ and can make between $30\,000$ to $240\,000$ measurements per second. Therefore, point clouds with a wider view can be created by rotating the sensor along one or more axes. If the sensor is moving at high speeds relative to the scanning motion, it is necessary to correct for this motion to avoid distorted point clouds.

When the scanning happens along a single axis, the result is a 2D "scan line", a series of range measurements made along a (possibly $360°$) arc. Such single scan-line measurements have been commonly used for obstacle detection and 2D mapping in ground robots. To create fully 3D point clouds, some lidar rigs, such as the one shown in Figure 2.3b, consist of stacked lidar sensors that simultaneously capture scan lines that, when combined, form a 3D point cloud. Another option is to rotate the lidar along two axes simultaneously; for example, the lidar in Figure 2.3a can operate in a "nodding" mode, capturing scan lines at various angles (Figure 2.2a). It is also common to use translational motion in a direction approximately orthogonal to the plane spanned by each scan line, which is usually known as a "push-broom" configuration (Figure 2.2b). Finally, some newer types of lidar sensors use a 2D array of sensors — like cameras — instead of a single sensor, allowing it to capture 3D point clouds with no moving parts. Some versions of these sensors are known as time-of-flight sensors.

Lidar has several advantages that have made it extremely useful in robotic applications. It is capable of capturing 3D measurements with a combination of accuracy, range and speed that is still out of reach for other methods; for example, the lidar sensor on the aerial platform of chapter 3 (Figure 2.3a) features sub-centimeter accuracy up to range of $200\,\mathrm{m}$. Moreover, its active nature

[4]There are several different variations of this term, including LiDAR, LIDAR, and LADAR (from LAser Detection And Ranging). Following the recommendation of Deering and Stoker [43], we adopt "lidar", which is consistent with similar abbreviations such as *sonar* and *radar*.

(a) (b)

Figure 2.2: Lidar scanning configurations. (a): Nodding lidar configuration. (b) "Push-broom" configuration.

makes it practically invariant to ambient lighting conditions, including the absence of light. The more powerful lidar sensors, such as the one used in chapter 3, are also able to partially penetrate vegetation and other obscurants, a useful capability in certain applications.

On the other hand, lidar sensors have certain downsides. They are relatively expensive, large, and heavy compared to sensors such as cameras. The latter two factors, in particular, are obstacles to their deployment in some robots, such as the micro-aerial vehicle we use in chapter 6. Lidar measurements are also relatively spatially sparse compared to those of high-resolution cameras, and in most sensors they lack visible-spectrum information such as color, which can be especially useful for semantic classification. For this reason, we supplement lidar measurements in camera imagery in chapter 5. Another issue with lidar is that it often yields incorrect or incomplete measurements for non-lambertian surfaces, such as transparent and specular surfaces. Examples include glass, water, mirrors, etc. Figure 2.4 shows an example captured from our aerial platform, where lidar fails to capture the stream in the center of the scene. While this thesis does not deal with this problem, using semantic inference to mitigate this issue — especially with multiple modalities, as in chapter 5 — is an interesting avenue for further research.

Stereo and structured lighting Structured lighting sensors and stereo image sensors both operate on the principle of triangulation of a signal from two (or more) viewpoints. In stereo sensors, this triangulation is performed by comparing image pairs captured by cameras in a binocular configuration. By matching the 2D projections of points that are visible in both images of the pair, the distance of each point to the camera can be inferred using simple geometry. If this inference is done for every pixel in the image array, *i.e.*, dense stereo, the result can be seen as a depth image which can be readily converted into a point cloud. The custom stereo rig used in the Micro-Aerial Vehicle (MAV) of chapter 6 is shown in Figure 2.5a, where the leftmost and rightmost cameras form a stereo pair.

Structured light sensors, unlike stereo, are active sensors. In structured lighting, one of the cameras is replaced by a projector that projectors a structured light pattern. The pattern is usually outside the visible spectrum, but visible to the remaining camera. By matching the structures of the pattern to their corresponding origin in the projector, structured light can triangulate the position of points using similar geometric principles to those used by stereo. The output, as in

(a)

(b)

(c)

Figure 2.3: Lidar sensors used in this thesis. (a) The lidar and nodding enclosure of the aerial platform used in chapter 3. (b) The lidar of the ATV platform used in chapter 5. (c) Time-of-flight sensor used for live system demonstration in chapter 4.

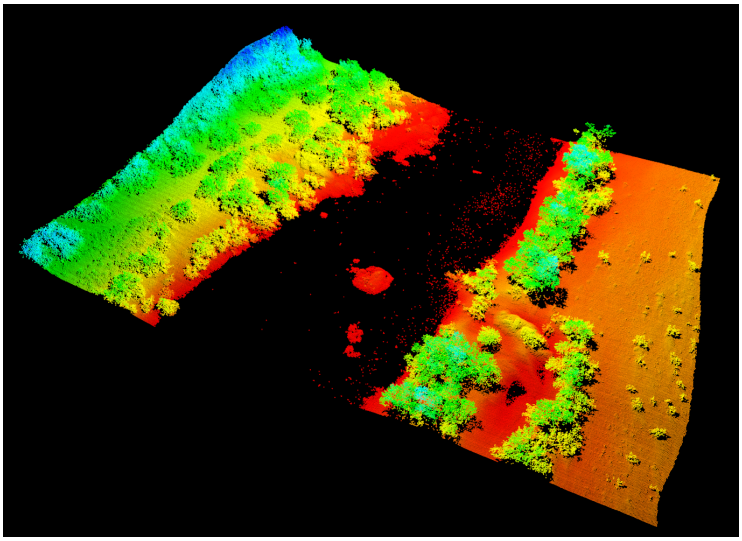


Figure 2.4: Example of lidar failing to capture a non-diffuse surface. The stream in the middle of the image is largely invisible to lidar.

(a) (b)

Figure 2.5: (a): Stereo sensing rig used in MAV from chapter 6. The leftmost and rightmost camera are used as a stereo pair. (b): Structured lighting sensor similar to the sensor used for the RGBD dataset [129] evaluated in chapter 4.

stereo, is a depth image. Inexpensive, mass-produced structured lighting sensors have recently gained wide adoption in indoor robotics and other applications. One such sensor (Figure 2.5b) was used for to capture the the experiments of chapter 4. Since commonly used structured lighting sensors usually capture RGB images along with depth (D) images, they are also known as RGBD sensors.

Stereo sensors and structured lighting sensors have different trade-offs. In stereo sensing, the main difficulty is finding pixel correspondences between the two images in the stereo pair; this is not only a computationally expensive process, but it is error prone, particularly in images that have a repetitive structure or that lack texture. Structured lighting sensors avoid the need for elaborate matching algorithms by using a pattern that is easy to detect and has a known structure. Therefore, structured lighting sensors are usually faster and more accurate. On the other hand, they are less robust to conditions which affect visibility of the structured pattern. For example, many structured lighting sensors use patterns in the infrared spectrum, which cannot be detected reliably under sunlight. Moreover, their range is usually limited by the intensity of the projector, and like lidar, they are adversely affected by non-diffusive surfaces.

Compared to lidar, both stereo and structured lighting have the important advantage of being comparatively inexpensive, small, and lightweight. However, they are also much less accurate and noisy. Their accuracy varies widely according to several factors, including their baseline (distance between the two cameras in the stereo pair or the projector and camera), the sensor resolution, algorithmic quality, the quality of the calibration, and distance from the sensor. The latter, in particular, is highly significant; due to the geometry of triangulation, range measurement precision decreases quadratically with distance. This limits the usefulness of these sensors for long-range measurements; for example, with the stereo sensor used in chapter 6, measurements beyond $20\,\mathrm{m}$ exhibited a standard deviation of more than two meters, despite using a relatively wide baseline configuration.

2.1.3 3D localization and mapping

So far, we have discussed sensing systems that allow the robot to estimate its pose and to capture point clouds of its environment, which brings us close to satisfying the requirements laid out in the beginning of this section. However, this sense data is not sufficient by itself to create consistent global 3D map. The range sensors we have described provide local measurements in an egocentric frame; as the robot moves around the world, measurements from these sensors must be integrated, taking into account the robot's motion, to create a coherent global representation of its surroundings, *i.e.*, a *map*. In many cases, the robot's own pose in the map will be estimated at the same time as the map itself, which is often known as localization; in this case, the task is usually referred to as Simultaneous Localization and Mapping (SLAM).

Localization, mapping and SLAM have been some of the most active areas in robotics in the last three decades. A comprehensive survey of the literature on this problem is out of scope; see Thrun, Burgard, and Fox [170] and Barfoot [6] for introductions to state estimation and SLAM from a robotics perspective, and see Cadena et al. [21] for a survey of more recent work on SLAM.

If we assume that the data from the sensors described in the last section is perfectly accurate (or nearly so), then a SLAM capable of fulfilling our spatial perception needs becomes a relatively simple problem. Localization in a global, geocentric frame can be performed by fusing GPS and

IMU measurements. Then, given accurate and instantaneous localization estimates, the egocentric 3D measurements from a range sensor can be transformed into a global frame. Accumulating these measurements over time then leads to a global 3D map.

Naturally, the assumption of perfect sensing is unrealistic, but given the high quality achievable by modern sensors, the method we have we have just outlined constitutes an effective starting point, and in fact is the basis for the localization and mapping systems used by the robots in this thesis.

One of the main issues with this baseline method is that it requires very precise pose estimation in order to transform 3D measurements in the egocentric frame to the global frame, as small perturbations in the pose (especially in attitude) can result in large changes to the transformed 3D measurements.

We note that errors in this case can stem not just from inaccuracy in pose estimation *per se*, but also time synchronization issues; for a mobile robot, using a pose estimate from two seconds ago to transform a point cloud captured one second ago will usually lead to highly noisy maps. Moreover, when range sensors move at high speed — because of a scanning motion, as used for lidar, or because the vehicle itself is moving at high speed, or both — it is usually insufficient to treat point clouds as a rigid constellation, and it is necessary to transform points individually to compensate for sensor motion during data capture [13, 189].

Aside from synchronization issues, there are many reasons INS may yield poor localization results, especially with lower-grade GPS and IMU sensors. For GPS, depending on the number of visible satellites and other factors, it is not uncommon for position estimates to fluctuate by several meters, especially along the vertical (height) axis. While the fluctuations can be smoothed with IMU (and possibly barometer) measurements, these sensors can introduce errors of their own, due to rapid drift accumulation and noisy measurements (*e.g.*, under vibration and jerky motions). Moreover, neither GPS or INS correct for drift in yaw, and the magnetometers used for this purpose can be affected by magnetic fields from various sources other than the Earth.

To improve the quality of the estimated poses, a common approach is to incorporate additional information, depending on the application. For wheeled vehicles, wheel odometry is a common sensor to fuse with IMU and GPS to improve motion estimation; the ATV in chapter 5 incorporates wheel odometry in its state estimation, alongside other cues.

A more generally applicable source for pose estimation is visual data from monocular cameras, stereo cameras, or RGBD sensors. Visual odometry [131, 95] infers relative motion over time by matching visual features between consecutive images; when a source of depth for these visual features is available (*e.g.*, via stereo or RGBD sensing), metrically scaled motion can be recovered. Steady improvements in this area over the years have resulted in increasingly accurate and robust systems for monocular SLAM [53, 127] as well as RGBD SLAM [130]. For robotic deployment, incorporation of IMU data is also useful, and Visual-inertial odometry or VIO has become an active research area [12, 141]. The MAV from chapter 6 uses visual odometry from stereo pairs, and fuses these estimates with data from GPS, IMU, and altimeter in a pose-graph based framework Song, Nuske, and Scherer [164].

Lidar can also be used for odometry by matching consecutive point clouds to infer motion; a variation of the method proposed in Zhang and Singh [189] was used in the autonomous ATV from chapter 5.

2.1.4 3D map representations and temporal fusion

When working correctly, the sensors and methods that we have described in the last two sections can provide our robots with a stream of global pose estimates and point clouds in a consistent global frame. However, it is usually impractical to use this point cloud data directly as a map.

A map is a model of the environment built from past observations. There are many common ways to build this model; from an engineering perspective, the choice of map representation is guided by a combination of factors, including fit with the robot's sensing and mapping capabilities, the ability to capture mission-relevant aspects of the environment, compatibility with the assumptions made by the robot's planning systems, and computational constraints.

Many different types of map representations have been proposed; see [18] for a survey. Below, we discuss the spatial map representations used in this thesis, and remark on how they can be used for temporal fusion of spatial data.

Point clouds

A point cloud map is a collection of three-dimensional points with known coordinates in a global frame. While many point clouds are created in an organized way (*e.g.*, an ordering given by the lidar scanline to which they belong), this structure is often ignored or unavailable, in which case the points are simply considered as an unordered, *i.e.*, unorganized point set.

Point clouds are a natural representation to use in many cases, such as when the data originates from a range sensor, or when the points are triangulated from features in Structure-from-Motion algorithms. In their raw form, they require little to no extra processing after the mapping process and can fully preserve the 3D data captured by the sensors, including reflectance, color, viewpoint, etc.

A disadvantage of point clouds is that large point clouds can become hard to manage computationally. Many common operations, such as neighborhood queries, are superlinear in the number of points, unless indexing data structures are used. Even then, with lidar or structured sensors capable of producing thousands of points per second, memory usage may become intractable.

Another disadvantage of point clouds is that they do not explicitly represent occupancy information of the environment; while each point provides evidence that the location of the point is occupied by a surface, there is an absence of evidence regarding locations with no points. However, if the viewpoint of origin of each point is known, then this information can be inferred, as we discuss below.

In this thesis, we use point clouds as the primary way to represent the raw data, as they are the natural way to represent data from lidar, and are also well suited to represent data from other 3D sensors. However, for semantic inference and mapping, we convert these point clouds into volumetric or height-based representations, which have better temporal scalability characteristics.

For temporal fusion purposes, point clouds have the advantage of providing a trivial solution in the form of storing all points seen in the past. However, in many cases this can result in impractically high storage and processing requirements over time. A simple and often used alternative is to store a subset of the past point cloud data, for example by keeping a bounded queue with the most recent points.

Depth maps

Depth maps are 2D arrays mapping pixels to depth in a sensor frame. These are a natural way to represent data from stereo cameras and structured light sensors.

As depth maps are inherently sensor-centric, for purposes of temporal fusion it is common to convert them into point clouds or integrate them in a volumetric representation. Another alternative is to simply store a past history of depth maps, along with their respective sensor pose [50].

Volumetric

Volumetric representations represent 3D attributes of the 3D world as a function of 3D coordinates, usually discretized into a uniform lattice of cells or *voxels*. Each voxel represents a spatial attribute, such as the probability of occupancy or signed distance to the nearest surface. In addition, voxels may be associated with extra metadata such as semantic labels or local shape features.

In this work, we use this representation extensively, as it provides an effective way to fuse multiple spatial measurements and represent uncertainty about the state of each spatial location [122, 41, 169]. Moreover, it is well suited for use with convolutional neural networks, as described in chapter 4.

On the other hand, volumetric representations have some disadvantages. The quantization process entails a loss of spatial information, and choosing a suitable voxel resolution can be challenging. As the resolution increases, more spatial detail is preserved, but computational and memory costs increase cubically and can become intractable.

Height maps

Height maps, also known as elevation maps, represent the height of surfaces as a function of 2D coordinates. Like volumetric maps, they represent space with a uniform lattice, but only in two dimensions. Each cell in a height map stores its estimated height (and optional metadata) in some reference frame; thus, they preserve a limited amount of 3D information, and are also referred to as 2.5D maps[5].

Like volumetric maps, height maps can be used to fuse spatial measurements [58, 61], albeit with an inherent loss of information caused by assuming only a single elevation value for every grid cell. The upside is a much lower computational cost, as the number of cells only increases quadratically with resolution. This trade-off is especially attractive for scenarios where the map to be modeled can be well-approximated by 2.5D representations. We use height maps to model surfaces in chapters 3, 5 and chapter 6.

2.2 Tools for semantic perception

Machine Learning (ML) is a key ingredient for our approach to semantic mapping. We aim to interpret the environment in terms of semantic categories such as *vegetation*, *ground*, *car*, etc., using the robot's sensors. As it is usually impossible to describe these concepts and their relation to

[5]Depth maps are also 2.5D maps, where each cell represents depth rather than height.

sense-data in terms of *a priori* rules or mathematical models, approaches that learn from data have become indispensable for tasks involving semantic analysis of sensor data. Within the literature in computer vision, robotics, and ML, there is a vast variety of ways to frame the problem of "semantic" understanding of the environment from data. For lack of a better term, here we will use *semantic perception* as an umbrella term to refer to these tasks. In this section, we identify the tasks that are most relevant to the work in this thesis, and how they relate to semantic mapping. We also summarize state-of–of-the–the-art approaches in each of these tasks.

2.2.1 Tasks in semantic perception

The two main tasks in semantic perception that are relevant to this thesis are *classification* and *semantic segmentation*.

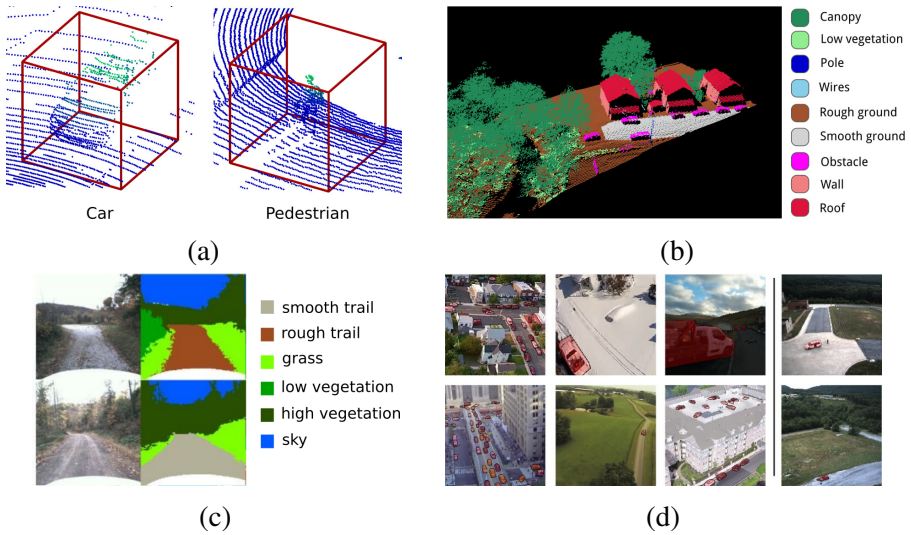


Figure 2.6: Examples of classification and semantic segmentation tasks that appear in this thesis. (a) Point cloud classification into one of 14 different categories relevant for urban driving [49] (chapter 4). (b) Multiclass semantic segmentation of aerial point clouds (chapter 3). (c) Multiclass Semantic segmentation of trail scenes (chapter 5). (d) Binary semantic segmentation of aerial (car/non-car) (chapter 6).

In *classification*, also known as *category recognition*, the goal is to associate a label from a set of K predefined classes (categories) to a given input — such as an image, a point cloud, or a volumetric input — as a whole. This is the most well-known task in semantic perception, not only because of its usefulness in and of itself, but also because many other tasks can be formulated in terms of classification. In all the classification tasks we will encounter in this thesis, we assume that there is a single class for each input. However, a common variation of this task — seen, for example, in the ImageNet benchmark [45] — is to allow multiple classes for each input, which is useful for problems with nonexclusive categories or inputs in which objects of different classes are visible. In this case, the task can also be seen as consisting of K binary classification tasks.

In *semantic segmentation*, the goal is to assign a label from a set of predefined classes to each location in the given input. For images, the "locations" are pixels; for point clouds, the

locations are points; for volumetric inputs, the locations are voxels. While not as well-studied as classification, this task has seen a sharp increase in interest over the last decade, given its wider range of applications.

A task that we do not address in this thesis, but is closely related to the above, is *object detection*, in which the goal is to infer the presence and location of instances of each class of interest. The key difference of object detection with semantic segmentation, for the purposes of this thesis, is that in detection the basic units of prediction and evaluation are individual localized instances (usually described by axis-aligned bounding boxes), whereas in segmentation the units are the dense locations (*i.e.*, pixels, points, or voxels)[6].

However, in this thesis we do use the term *detection* in a broader sense for certain tasks in which we aim to obtain localized predictions for the presence of a target class, but do not explicitly distinguish between instances of the same class. In this case, the detection task is equivalent to semantic segmentation with two classes. For example, in chapter 4 we refer to the task of finding landing zones for an autonomous helicopter as landing zone detection.

Naturally, the tasks are closely related. For example, semantic segmentation can be seen as classifying individual locations in the input, and many approaches to segmentation can be seen as implementing some variation of this approach. Likewise, in many methods for detection, classification is applied to multiple candidate bounding boxes (regions of interest) of the input.

2.2.2 Current approaches to 2D semantic perception

In the last few years, Convolutional Neural Networks (CNNs), and more broadly, *deep learning*, have become ubiquitous in current approaches to semantic perception in computer vision. Despite their recent notoriety, they have a decades-long history [105, 64]; see [70] for an overview.

The key feature of deep learning approaches, as the name suggests, is learning representations from data at successively deeper levels of abstraction. In CNNs, this is realized by stacking layers of learned convolutional filters Figure 2.7. By comparison, until recently, the dominant approach was to apply hand-engineered feature extractors to the data, and use these features as input to a learned classifier; in deep learning and CNNs, the features and classifiers are learned jointly from the data.

CNNs rapidly rose in popularity for the task of object recognition following the success of the *AlexNet* CNN architecture [98] in the 2012 ImageNet benchmark, and since have become the *de facto* standard approach to this problem. Since the development of AlexNet, a vast number of new CNN architectures have been developed, with popular examples including VGG [161] and ResNet [147].

CNNs are also part of the current state of the art for object detection. One line of approach divides the problem into two subproblems: the problem of proposing regions that may have objects, and the problem of classifying regions. In [67] the former problem is addressed with a hand-engineered method [174], and the latter is solved with an AlexNet-style architecture. In [147], both problems are solved jointly with a single network. Another line of approach poses the problem as regressing bounding box coordinates [167, 145]. Recent work also has sought to improve accuracy

[6] In recent years, the distinction between detection and segmentation has blurred, as tasks that combine aspects of both have emerged, such as instance segmentation [79] and panoptic segmentation [94].

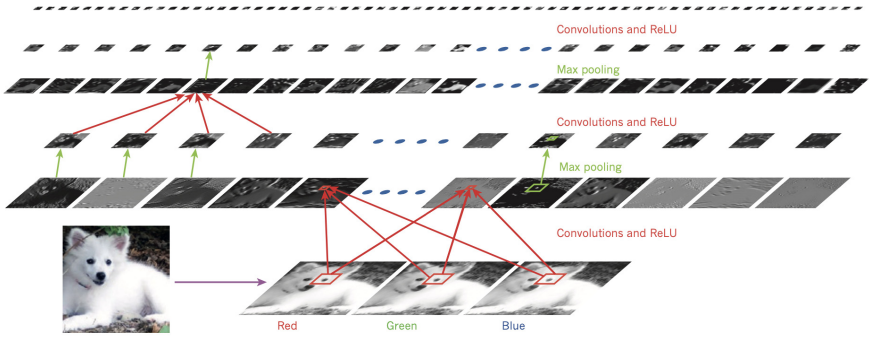


Figure 2.7: Example of deep representation learning in CNNs. Figure reproduced with permission from LeCun, Bengio, and Hinton [106].

by modeling context, usually by combining CNNs with graphical models [192, 110].

For semantic segmentation, CNNs have also become an integral part of state-of-the-art pipelines. Some approaches work by segmenting the image and extracting CNN-based features from segments [123, 59]. A recent line of alternatives, which we adopt, avoid segmentation by directly predicting dense pixelwise labelings [113, 51, 188, 5]. These approaches take advantage of the convolutional nature of the task to perform efficient feed-forward classification. Foregoing some of this efficiency for higher accuracy, many approaches have combined CNNs with probabilistic structured output approaches [30, 109]. On the other hand, there has also been recent work in accelerating inference, with some penalty on accuracy; examples we use in this thesis include [137] and [177].

2.2.3 Current approaches to 3D semantic perception

Much of the work in 3D semantic perception has mirrored developments in 2D perception, but the adoption of deep learning has been slower. One likely reason is the greater diversity in possible input representations used for 3D tasks; whereas in image data, pixels are a nearly universal common denominator, in 3D different algorithms depth maps, point clouds, meshes and volumetric representations.

A large body of work has been developed around depth maps, largely due to the popularity of structured-lighting RGBD sensors, and to a lesser extent, stereo sensors. Much of this work directly applies the approaches developed for RGB data to depth data. This representation is also amenable to adaptation of CNN approaches used for image data.

For recognition, [163] use a "recursive" neural network; [155] colorize depth images for use with a standard CNN. For the detection of graspable points [107] use a sliding window approach with a neural network cascade. [74] use a slightly more sophisticated encoding of depth.

For segmentation, [113] and [19] use depth images with CNNs for indoor semantic segmentation, and [112] do the same for an outdoors urban scenario.

There is also much work using a point cloud representation. To our knowledge, no work with point clouds has adopted Deep Learning yet, and most use hand-engineered features from point clouds. Examples include [69] for recognition, and [126, 159, 2] in per-point semantic segmentation.

There are also various recent works for semantic analysis in 3D using volumetric representations, including our own; we review them in chapter 4.

2.2.4 Relation to semantic mapping and our thesis

In general, we can see the semantic perception tasks we have described above as a subset of the semantic mapping task.

For semantic perception tasks with images, the difference with semantic mapping is clear; if we are to build a semantic map with a 3D representation of the environment, then labeled pixels or bounding boxes in 2D are insufficient, and additional steps are required to incorporate them in a semantic map.

For the case of 3D semantic perception tasks, the distinction is more subtle. For 3D object recognition, the problem scope is more limited in comparison to semantic mapping. Object recognition assigns a label to a given input (*e.g.*, a point cloud), but does not address how to select this input or use the output label. On the other hand, for 3D detection and semantic segmentation, their tasks can be very close to what we consider semantic mapping. Often, there are differences in the application, which are reflected in the different assumptions and engineering decisions made in each case. For example, many approaches in 3D semantic segmentation assume the whole point cloud to be segmented is available at once, and are too computationally intensive for real-time usage. In contrast, practical semantic mapping approaches require online, real-time operation with data received incrementally.

In this thesis, we will encounter examples of the three semantic perception tasks defined above, and we will build on the approaches addressing them. In chapter 3, we will perform voxel-based semantic segmentation of point clouds. In chapter 4, our main focus is object recognition for 3D data, but we adapt the object recognition approach to perform a type of detection. Finally, semantic segmentation, primarily in 2D, is a prominent part of chapter 5 and chapter 6.

2.2.5 A note on recent progress in semantic perception

The state of the art for semantic perception tasks has vastly changed in the years since we began the research described in this thesis. In 2014, research in deep learning with 3D point clouds and its applications to robotic tasks was nearly nonexistent. Since then, many factors, such as the surge of interest from industry in self-driving technology, and more generally, the cambrian explosion of research in the fields of AI and robotics, have led to the publication of a large body of work that is relevant to the work in this thesis that we have not covered here. While we are unable to cover this literature in detail, in the last sections of chapter 4, chapter 5 and chapter 6 we briefly discuss selected recent works that build on our own, address limitations of our approaches, or are otherwise relevant.

Chapter 3

Ground surface-aware semantic mapping

> Keep your feet on the ground and your thoughts at lofty heights.
>
> Peace Pilgrim
> *Her Life and Work in Her Own Words*

Recent technological advances have made the prospect of autonomous aerial vehicles operating in the wild a question of *when*, rather than *if*. With recent advances in sensing and mapping technology, aerial vehicles can quickly capture detailed point clouds of their surroundings. By creating semantic maps in real-time from these point clouds, we can provide these aerial vehicles with valuable information for safe navigation and landing in human-populated areas.

In particular, these semantic maps are most useful for point clouds captured at low heights, ranging from $10\,\mathrm{m}$ to $50\,\mathrm{m}$. However, this is a relatively unexplored regime for semantic point cloud segmentation, where most prior approaches have been designed for ground-level data or high-altitude aerial surveys. One of the previously unaddressed challenges in this scenario is the accurate estimation of the ground surface height, a critical cue for semantic inference and aerial planning. In addition, prior work in aerial point cloud segmentation has focused on offline batch inference, rather than real-time, incremental operation.

To address these challenges, we propose a system featuring two innovations relative to the state of the art in semantic point cloud segmentation. First, we propose a method to infer the ground surface in the presence of occlusions, and use the inferred surface to inform semantic inference. Second, we achieve real-time operation with grid-based representation encoding sufficient statistics for point clouds that enables efficient feature extraction and incremental updates.

While the ideas in our approach are potentially applicable for various kinds of real-time semantic mapping tasks with aerial point cloud data, in this chapter we will evaluate our system in the context our motivating application, which is to provide a full-sized helicopter with semantic awareness of classes relevant to navigation and landing. We assemble a dataset of manually labeled point clouds captured by our platform, and show the benefits of our ground surface estimation step for semantic inference. In addition, we demonstrate real-time operation of our pipeline from streaming point cloud data. In the next section, we will provide further context on this task and our platform.

3.1 Semantic awareness for an autonomous helicopter

Helicopters and other rotorcraft are widely used in transportation of people and cargo, search and rescue, firefighting, and aerial surveillance, among other applications. Currently, these vehicles need to be operated by highly trained pilots, which in many situations may be in short supply or would be put at risk by piloting the vehicle. This has motivated research aimed at enabling rotorcraft to navigate, avoid obstacles, and land with minimal or no human intervention. As part of this broader research effort, our research group participated in the development of an autonomous full-scale helicopter for cargo transportation and medical evacuation [135]. The work in this chapter was spurred by challenges encountered in the development of this system. The vehicle used for the system (shown in Figure 3.1), is equipped with a custom sensor suite including a scanning lidar, GPS-INS and cameras, which are used to create detailed point cloud maps of its surroundings.

Figure 3.1: The Unmanned Little Bird platform performing an autonomous landing

While effective for geometric path planning and obstacle avoidance, the point cloud data lacks information that is useful for safe navigation and landing in cluttered, unstructured environments. For example, when planning trajectories it is desirable to give more clearance to buildings than tree canopies, as a collision with the former is more dangerous for both the vehicle and the building inhabitants (Figure 3.2). However, if space as seen merely as occupied or unoccupied, the planning system is unable to incorporate this preference.

Even if basic geometric properties of point cloud data, such as local planarity, are taken into account, these are often insufficient to discriminate the classes of interest. For example, building roofs and ground planes both are locally planar, but the latter is usually preferable for landing.

Furthermore, given that lidar measurements are affected by noise, sparsity and occlusion, the geometric information in point clouds in itself is not completely reliable:

- Specular reflections and noisy returns should be discarded
- Wires, given their small surface area, may only be partially seen by lidar

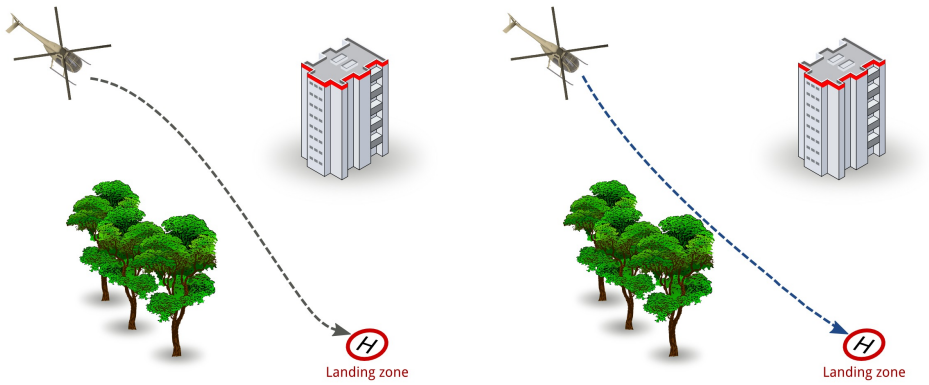


Figure 3.2: Semantic awareness of the environment allows the helicopter to incorporate the preference of giving more clearance to buildings than tree when planning its flight path.

- Ground surfaces may be occluded, or receive sparse coverage

From these observations, we conclude that a system to attribute localized semantic information — in the form of hand-chosen relevant categories — can allow more deliberative and adaptive decision making for the vehicle.

Given the intended application, the system must also meet certain requirements:

- Low enough latency to be used in real-time planning
- Incremental operation from streaming point cloud data, captured from an agile moving platform
- Robustness to noise, sparsity and occlusion

In the next section, we will survey related work that addresses tasks similar to ours, and in many cases, provided useful insights for our work. The rest of this chapter describes the system we designed and implemented to meet our goals.

3.2 Related work

Semantic segmentation of point clouds The first task in our problem statement, predicting semantic categories to points in a point cloud, corresponds to the *semantic segmentation of point clouds* task discussed in section 2.2.

Our system was heavily influenced by the state of the art in semantic segmentation of point clouds at the time of its development, which showed the feasibility of using machine learning models to reliably discriminate among various classes of interest solely from lidar data.

Most of these systems employ a pipeline influenced by contemporaneous work in the area of computer vision, in which various hand-engineered features are extracted for each point (or local subregion) to label, and labels are inferred from these features by classifiers trained with labelled data. In this work we follow this pattern. In particular, our shape point clouds include spectral shape features [103, 93], various height and lidar waveform features [29], and features based on pass-through estimation [181]. However, we complement these with our own height

features based on the inferred ground surface. We also use random forests [14] as the classifier, a common choice for similar systems [29, 23, 159].

In addition, we depart from most previous work that computes features in batch, usually with exhaustive per-point neighborhood queries. Instead, we compute these features from incrementally computed point cloud statistics which are maintained in a voxel data structure. The data structure is inspired partly by that of Hu et al. [83], but unlike this work, we use the data structure to maintain sufficient statistics instead of individual points.

Ground filtering The second task in our problem statement is predicting ground surface heights. There is a large body of work in robotics devoted to the problem of modeling the geometry terrain around the robot with different representations and methods. A representative and widely used approach is Fankhauser et al. [58], which uses a grid-based 2.5D elevation map with per-cell height and uncertainty estimation. These approaches generally consider the "terrain" to be all surfaces visible to the robot, and do not attempt to distinguish the bare earth terrain from other objects. While the outputs of these methods may be further analyzed for semantic analysis, by themselves they are insufficient.

An approach that is close to ours is Wellington [181], which implements a system to estimate potentially occluded ground surfaces from lidar and image data using a generative model. Like our approach, it is intended for online incremental use and is designed to deal with heavy occlusion. However, as it is designed for an off-road vehicle, it makes assumptions that are better suited for inference over small areas compared to the ones we wish to cover. Their 3D features are relatively limited in the spatial context they capture, and they rely on local smoothness assumptions for their probabilistic model to propagate context. This probabilistic model uses Monte Carlo sampling for inference, which is computationally expensive and suffers from unpredictable mixing times. In contrast, we use a simpler model with a more efficient and predictable inference process.

There is also relevant work in the areas of remote sensing and aerial lidar analysis. Our ground surface prediction can be seen as a type of *Digital Terrain Model* (DTM) or *Digital Elevation Model* (DEM) in the Geographic Information Systems (GIS) literature[1].

See Meng, Currit, and Zhao [121] for a survey, and Sithole and Vosselman [162] for a comparison of the most popular methods. While these methods address a similar task to our ground surface prediction, given their origin in GIS, they assume the lidar data is extracted from a nadir angle and high altitudes. In this regime, occlusions and noise are less significant than in our data, which may be captured from oblique angles and much closer to the ground ($50\,\mathrm{m}$ or less). These methods are also not designed for incremental or real-time use, as these are not critical for GIS.

3.3 Approach

At a high level, we formulate the problem as follows:

> *Given a stream of 3D point clouds, vehicle poses, and 3D regions of interest (all of which are temporally synchronized and registered to a common inertial frame),*

[1]These terms are often used interchangeably; however, if a distinction is made, DEMs represent the elevation of all objects in an area, including buildings, whereas DTMs represent the elevation of the bare earth, which is closer to our task.

construct and update a 3D semantic map with 1) a predicted semantic category for each visible point within the region of interest, and 2) a 2.5D height map with a predicted predicted ground surface height for every location in the region of interest.

In this section, we will first discuss in subsection 3.3.1 in further detail the assumptions we make regarding the inputs and outputs for our problem. In the remaining sections, we will describe our proposed approach, outlined in Figure 3.3.

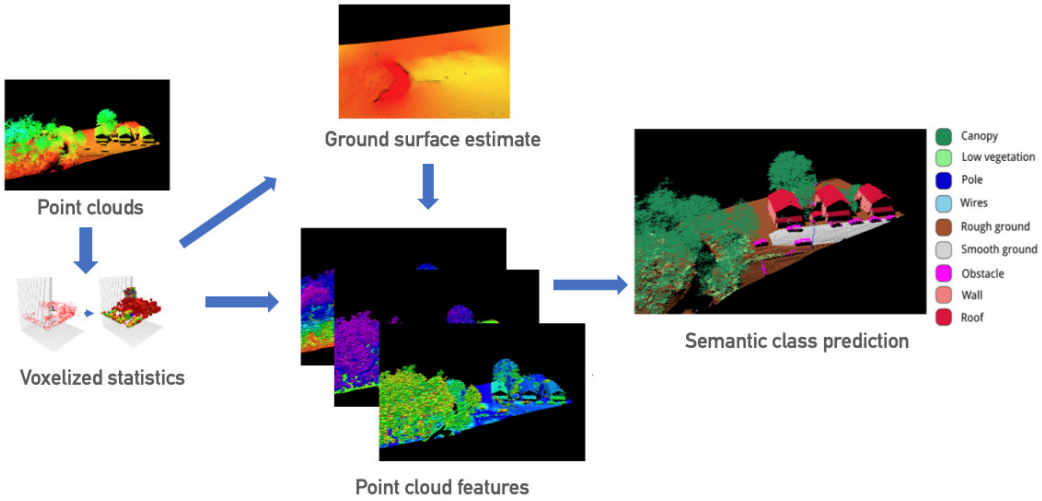


Figure 3.3: Ground surface-aware semantic segmentation system overview

3.3.1 Problem setup

Inputs: 3D point clouds, vehicle poses, and regions of interest (ROIs)

Our system only uses sensor data originating from the vehicle's lidar and GPS-INS. We do not use the data from these sensors directly; instead, we rely on the state estimation and perception modules of our platform, which fuse this data to create a temporally synchronized stream of 3D point clouds and vehicle poses, expressed in a common inertial frame. We use these outputs as provided; we do not explicitly attempt to model or account for errors in the registration or state estimation. However, our system performs temporal and local spatial smoothing that mitigates the effect of small amounts of noise and errors in these variables.

The lidar is a customized survey-grade sensor with a single-line scanner mounted on an actively controlled nodding platform (Figure 2.3a). The range, field of view and points-per-second of the measurements depend on the sensor's configuration. For most of the data we use in this chapter, the sensor is configured as a "push-broom" (Figure 2.2b) scanner with a moderate ($0°$-$30°$) off-nadir pointing angle and a horizontal field of view of approximately $60°$. In this configuration, the sensor has a range of $10\,\mathrm{m}$ to approximately $200\,\mathrm{m}$, with an accuracy of $\pm\ 10\,\mathrm{mm}$. The 3D point clouds are streamed at a rate of approximately $1\,\mathrm{Hz}$, with each point cloud having approximately $20\,000$ points. In addition to 3D position, the provided point clouds contain other per-point attributes such as surface reflectance, instantaneous origin of the lidar ray, and echo number.

As for ROIs, we assume that for any given moment in time, there is a bounded 3D Region Of Interest (ROI) to be mapped. This ROI can be static (either relative to a fixed global frame or a local vehicle-centric frame), or dynamically adjusted in a data-dependent way.

The motivations for this assumption are twofold. The first is that this allows us to simplify the problem by ignoring distant regions where point cloud data is likely to be extremely sparse, noisy, or absent altogether. The second is that this makes it easier to bound the computational resources (in memory and time) used by our solution, so that real-time constraints can be met. This assumption is not highly limiting, given that in general only a subset of the 3D space around the vehicle is relevant for planning.

In practice, we mainly use two ROI strategies. The first is a static ROI relative to a global frame, which is useful when there is a known (but possibly approximate) location for the surface to be scanned; an example is the landing zones in section 3.4.2.

The other main strategy, which is used in the qualitative results (section 3.4.2) we use is to dynamically adjust the horizontal offset of the ROI to track the robust centroid (in XY) of the incoming point clouds, and the vertical offset of the ROI to stay at a certain height relative to our last estimate of the inferred ground surface. We use this strategy because simpler strategies, such as maintaining the ROI at a fixed location relative to the vehicle frame, tend to discard large amounts of data given the variable heights at which the helicopter flies.

Outputs: 3D semantic map and 2.5D ground surface map

Each point in the map will be assigned a label from a finite set of of semantic categories specified *a priori*. We choose categories heuristically based on a qualitative assessment of their potential usefulness for planning and navigation, as well as whether the categories are identifiable from lidar data. The nine classes we predict are *ground*, *building roof*, *building wall*, *low vegetation*,*tree canopy*, *obstacle*, *pole*, *wire/power line*, and *paved ground*; see Table 3.2 for descriptions.

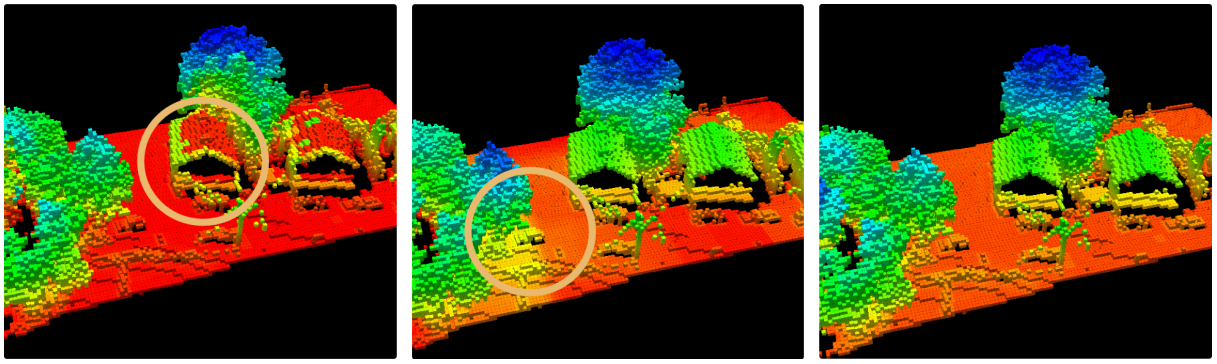
Our approach needs a dataset of hand-labeled point clouds with the chosen semantic categories, that is used to train the machine learning models employed by our approach and evaluate their accuracy. We describe our custom dataset in subsection 3.4.1.

At runtime, for each point in the incoming point cloud that falls within our ROI, we predict a distribution of our beliefs regarding its semantic category, considering all data seen in the past.

As an important detail, we note that the points are not labelled individually; rather, they are grouped in local subvolumes (*i.e.*, the *voxels* described below), each of which has a single label distribution that is inherited by all points within. Thus, the effective spatial resolution of our predictions is lower than that of the input point cloud. In some applications, it may be desirable to discard the original point clouds and only use the quantized representation given by the semantic map. In our case, we still use the original point cloud data for visualization and to maintain compatibility with other components that use point clouds as their input. In any case, in our representation, assigning a label to a point according to its voxel is a constant time operation that adds negligible overhead.

We also explicitly model the bare-earth ground surface, defined as the top soil or a thin layering (*e.g.*, asphalt or pavement) above it [162]. Whereas categories other than ground are only predicted for visible 3D points within the ROI, we predict a ground surface for every location in the ROI, whether or not there is a visible point at that location.

This is motivated by a combination of two factors: knowledge of the ground surface is highly useful for the task of finding landing zones, as well as accurately predicting semantic categories (Figure 3.4); and the fact that due to the vehicle's high vantage point, large portions of the ground surface are occluded.



(a) Height with minimum in $1\,\mathrm{m}$ radius.

(b) Height with minimum in $3\,\mathrm{m}$ radius.

(c) Height with our inferred ground surface.

Figure 3.4: Motivation for ground surface estimation. Each figure shows a voxelized point cloud colorized according to its *height* feature, using three different methods to estimate this feature. Voxels with the same color are estimated to be at the same height. In fig. (a), the height is estimated as the distance of each voxel along the z-axis to the lowest point in a $1\,\mathrm{m}$ radius. Note that the estimate is noisy for the encircled house roof, as it spans more than $1\,\mathrm{m}$. In fig. (b), the same estimate is performed with a $3\,\mathrm{m}$ radius. In this case, the estimate is accurate for the house roof, but inaccurate for the circled car, because the slope of the ground skews the estimate over large distances. In fig. (c), height is estimated relative to our inferred ground surface, and an accurate height estimate is recovered for both the house roof and the car. This shows how height computed relative to our inferred ground surface is a more useful feature for classification than height estimated with more naive methods.

We make the simplifying assumption that the ground surface can be modeled as a 2.5D elevation map, *i.e.*, a map where a height or elevation is assigned to any given 2D location within the ROI. This assumption facilitates the development of efficient algorithms for dense prediction and is sufficient for the environments where the vehicle operates.

In contrast to the situation with the other semantic categories, we do not assume that ground truth for this task is available, given the relative difficulty of obtaining this data or creating it manually.

3.3.2 System overview

Our system operates online and incrementally, updating the semantic map representation each time a point cloud is received from the sensor. In each update, the system performs a series of steps, outlined in Figure 3.3.

The "zeroth" step is preprocessing of the input data, where we heuristically eliminate spurious point cloud data from our inputs. Then, the point cloud is used to update a set of per-voxel

cumulative statistics describing the point cloud geometry and waveform characteristics. These statistics are maintained in a dense grid arrays for computational efficiency. We then use a first set of point cloud features and learned classifiers, along with an interpolation step, to estimate a 2.5D ground surface. The point cloud statistics and the ground surface are then used to extract further per-voxel point cloud features, including features that depend on the newly inferred ground surface. Finally, the point cloud features are given as input to a learned classifier to obtain per-voxel semantic class predictions.

The next sections provide further details on each of these steps.

3.3.3 Preprocessing

Due to various sensor-specific factors such as multipath scatter and occasional localization drift, the point clouds present spurious measurements. Many of these points are detectable by having an extremely low intensity or by having a physically implausible position in space, such as coming from behind the laser or extremely distant to neighboring points in the laser scanlines. We remove these points from the incoming point cloud with manually set thresholds before further processing.

Additionally, we require a minimum number of points to be accumulated in each voxel for it to be considered in the prediction process. Points that fall within voxels that have less than this number of points are assigned a special label of "noise" and otherwise ignored.

3.3.4 Sufficient statistics and map representations

We maintain a set of grid-based data structures that maintain a representation of past point cloud data, intermediate features, and per-cell label prediction beliefs. To this end, we store data in two ways: the first is a dense array of sufficient statistics for each grid cell in the ROI, and the second is a sparse table that only stores data for grid cells with visible points. The table contains point cloud features and a vector of predicted semantic label probabilities. For each of these, we have two versions: a 3D version, where each grid cell corresponds to a voxel, and a 2D version where each grid cell corresponds to a 2D area along the XY plane.

Dense array for sufficient statistics

As mentioned earlier, at any given moment in time we will have a 3D region of interest, which may be either come from a predefined relationship to the vehicle's pose, or computed dynamically according to the incoming point data. In either case, the ROI will be defined as an axis-aligned 3D bounding volume with an (time-varying) offset relative to a fixed frame. The dimensions of the bounding volume are static.

We partition this ROI into a regular 3D lattice, such that each cell, or *voxel*, can be indexed by an integer 3-tuple. The physical dimensions of each voxel are a predefined parameter; in practice, we use cubical voxels of size $0.25 \times 0.25 \times 0.25\,\mathrm{m}^3$. As the bounding volume has a fixed size, the number of cells remains fixed. Additionally, we create a parallel 2D grid along the XY plane, used to maintain features that do not vary with height.

The first step in the point cloud processing is to discard points for which the ray from the lidar does not intersect the ROI. We then quantize the original point cloud coordinates (x, y, z) into

integer coordinates (i, j, k) corresponding to a voxel in the grid map (Figure 3.5). This is done by a simple offset and truncated division. The first two coordinates, (i, j), index into the parallel 2D grid map.

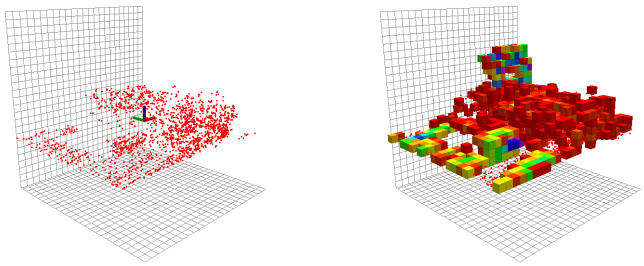


Figure 3.5: Our representation summarizes the incoming point clouds with a set of sufficient statistics computed for each 3D grid cell (voxel).

We maintain a running set of sufficient statistics for each grid cell (in 2D and 3D) as a dense 2D or 3D array. Using a dense array allows for constant time access and efficient traversal over subvolumes.

For scalar values, we maintain the sufficient statistics to compute their first two moments, mean and variance. For 3D points, we maintain the sufficient statistics to compute their first two spatial moments, *i.e.*, their centroid and 3×3 covariance matrix. In addition, we maintain the running number of observations n in each 2D grid cell or 3D voxel cell.

For each 2D grid cell, we maintain the sufficient statistics of the height z of each point that is quantized to the same grid cell. Note that here, the height z is an arbitrary global frame, which may be *e.g.*, relative to mean sea level, and not necessarily indicative of the height relative to the ground.

For each 3D voxel cell, we maintain the sufficient statistics for the point centroid and covariance (in 3D), the first two moments of the lidar range, the first moment of the lidar reflectance, the number of interior returns, and the number of rays that *pass through* or *hit* each voxel. For the ray calculations, we use a 3D line rasterization method [1], and we include all rays that intersect the ROI, even if they do not end inside it.

These statistics are sufficient to efficiently compute all the point cloud features that we use for classification of each voxel. For example, spectral features can be directly computed from the 3D covariance matrix; porosity can be directly computed from the number of hits and pass-throughs in each voxel; number of points below each voxel can be computed by simple addition of the number over a voxel column, etc. Therefore, there is no need to perform expensive spatial point queries to extract these features. Instead, most features can be computed directly from each grid cell or voxel, or in some cases, with a limited amount of traversal of grid or voxel neighborhoods.

We periodically update our grid map to track the 3D ROI as it moves over time. In the fixed 3D frame, this operation is a simple 3D translation. Because of this, we can efficiently update the dense arrays we use to store sufficient statistics without reallocating memory by virtually "reindexing" the arrays, in a 3D analog to circular arrays or ring buffers. This is the "scrolling"

operation.

Sparse table for features and label predictions

For each nonempty grid cell, we compute a vector of point cloud features (section 3.3.6) and a vector of predicted semantic label probabilities (section 3.3.6). Storing these in a dense 3D array would be onerous in terms of memory. Instead, we store them in a hash table keyed by a quantized 32-bit representation of the integer coordinates of each cell. To bound the memory footprint of this table, we periodically clear the least recently created entries.

3.3.5 Ground surface estimation

For ground surface estimation, there are two steps (Figure 3.6). In the first step, we detect a set of grid cells from the 2D grid map which we confidently consider as *ground* using a learned classifier. In the second step, interpolate the estimated ground surface over the rest of the 2D grid cells using an unsupervised Gaussian Markov Random Field.

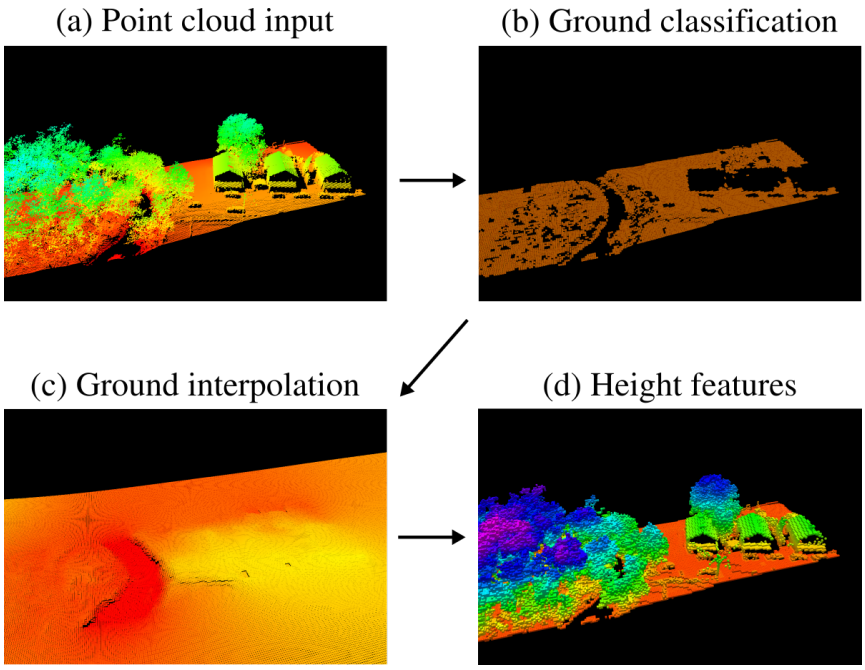


Figure 3.6: Steps in ground surface estimation. Given the point cloud ((a)), a set of confident ground surface cells is detected ((b)). These confident cells are used to interpolate (and extrapolate) a ground surface to the rest of our ROI ((c)). This ground surface serves as a source of robust height features for classification ((d)).

Ground cell classification

The goal of this step is to obtain a set of grid cells that are confidently labelled as *ground*, which serve as "control points" for the interpolation process in the next stage. To this end, we use an

decision forest classifier, similar but separate from the classifier used to predict the final semantic labels.

First, we extract a subset of the point cloud features described in section 3.3.6 for each nonempty cell. We then train a binary decision forest to predict whether each cell is *ground*, using our hand-labelled dataset. At run time, we make a binary decision on whether each nonempty cell is ground using a high confidence threshold, so as to avoid false positives affecting the result of interpolation.

Ground surface interpolation

In the first stage, we obtained a sparse set of grid cells predicted as belonging to the ground surface. We use the height measured in these grid cells to predict the unobserved ground surface height of the remaining cells in our ROI.

For this purpose, we use an autoregressive model on the grid lattice [9], a type of Gaussian Markov Random Field (GMRF) model [151]. In this model, the height of each cell, z_i, is assumed to be a noisy average of its neighbors:

$$z_i | \{z_j : j \neq i\} \sim \mathcal{N}\left(\frac{1}{n_i}\sum_{j \in \mathrm{N(i)}} z_j,\ \frac{1}{n_i\,\kappa}\right)$$

where n_i is the number of neighbors of cell z_i, and κ is an inverse variance parameter. As we use a 4-connected neighborhood, $n_i = 4$ (except at the borders, but as we discuss below, we avoid this case). This model can also be seen as solving the Laplace equation on the grid [82].

In this form, the joint distribution of the heights is normal, with a mean that can be assumed to be $\mathbf{0}$ and a precision matrix $\mathbf{Q}$:

$$Q_{ij} = \kappa \begin{cases} -1, & i \text{ is neighbor of } j \\ n_i, & \text{i = j,} \\ 0, & \text{otherwise} \end{cases}$$

The sparsity of $\mathbf{Q}$ can be used to efficiently infer the expected heights of the unobserved grid cells given the observed grid cells, among other tasks. We use an off-the-shelf sparse Cholesky solver from Eigen [73], which leads to a complexity of $O(n^{3/2})$ in the number of cells to interpolate.

The resulting model is simple to interpret and implement, as well as relatively efficient compared to alternatives such as Gaussian Processes. It is more robust to noise than simple alternatives such as nearest neighbor interpolation.

This simple model does have drawbacks. Its assumptions regarding conditional independence and stationarity lead to efficient inference and straightforward implementation, but make the model inexpressive and unsuitable for interpolation of complex 3D surfaces. However, this is rarely necessary for our scenario, as we operate in spatial scales for which the ground surface is relatively smooth and flat.

We note that we use the GMRF only for interpolation, not extrapolation, *i.e.*, only to infer the height of unobserved cells that are surrounded by observed cells. There are two reasons. First, in many cases, there is a large number of unobserved cells beyond the visibility frontier of the lidar, which can incur in a large computational cost for the GMRF inference. Second, we found the

behavior of common methods to deal with boundary conditions (*e.g.*, assuming a toroidal lattice) to be unsuitable for our problem. Instead, inspired by Lai et al. [102], we use a recursive median filter (RMF) for these cells. The RMF has linear complexity, and is less sensitive to outliers than other linear-time methods such as nearest neighbor interpolation.

One important drawback of the GMRF is that it is sensitive to outlier height measurements, such as those caused by spurious lidar returns, or false positives in the initial ground segmentation step. These measurements will potentially induce large errors that are propagated in the interpolation process. We reduce the number of outliers as much as possible by filtering the raw point cloud and using a conservative threshold for the initial ground segmentation step, but some errors are inevitable. Fortunately, due to the incremental nature of our method, these errors are usually transient and their effects disappear as the vehicle gathers lidar observations over time.

Nonetheless, even transient errors can cause issues in planning and navigation, and a more robust method is desirable. We investigated more robust MRF methods (*e.g.*, replacing the normal distribution with a fat-tailed distribution), but found the computational requirements of these versions to be unacceptably high, and leave this direction for future work.

3.3.6 Semantic label prediction

Point cloud features

As mentioned in section 3.2, we draw from various papers in the literature to build a rich set of features from our point clouds. There is a rich set of features described in the literature, several of which appear to have been reinvented more than once. During development, we experimented with various features described in [29, 158, 23, 93, 103, 62, 185, 156], among others. Besides the ground surface-aware features, we also devised some simple descriptors that to our knowledge, have not been described before in the literature. After excluding features with relatively high memory or processing requirements, we erred on the side of having an overcomplete feature set, as our classifier of choice has feature-selecting properties. Our features are summarized in Table 3.1. Per the table, they can be grouped in four categories:

Local shape These features describe the local shape of the cloud, as approximated by a Principal Component Analysis (PCA) of the points in the voxel. These features describe how well the local point cloud approximates a line or a plane (or neither), which is useful, *e.g.*, to discriminate walls, wires, etc. We also describe orientation of the normal and tangent vectors of the local plane relative to the direction of gravity (vertical) and relative to an arbitrary vector perpendicular to gravity (horizontal). These features help with horizontal surfaces (such as roofs or ground) as well as vertical (walls).

Height and rays For each voxel, we calculate various properties relating to the distribution of points above and below. This is useful, *e.g.*, for ground (which has few points below it) and canopies (which tend to have points below it). We measure this property at various neighborhoods around each voxel, as most surfaces will occlude the points directly beneath it. However, large objects (such as buildings) will occlude the ground beneath over large spatial extents; this motivates

the use of our inferred ground surface to compute this property, rather than the raw point cloud statistics.

We also compute various properties derived from the statistics of the rays formed from the lidar sensor to its corresponding point measurement, which conveys information not fully described by the points themselves. For example, inspired by [76] we compute how many rays pass above each voxel; this may help in discriminating ground surfaces (as many rays will pass above). Similarly, inspired by occupancy mapping [90] we compute for each voxel its "porosity", as the ratio of lidar rays that hit a given voxel versus rays that pass through, as *e.g.*, vegetation and wires tend to have high porosity compared to walls and the ground.

Lidar waveform While we do not have access to the full waveform, our lidar reports additional information for each point besides its range. It reports the measured reflectance for each pulse, as well as an index for each return. It is common for an outgoing lidar pulse to have multiple returns, and different objects tend to differ in how often they are reflected in the final return versus earlier (interior) returns (Figure 3.7). In particular, vegetation and other porous objects present a higher ratio of interior to final returns compared to solid surfaces.

Range Finally, we compute two features based directly on the range, *i.e.*, the distance of the lidar to each point at the time of its measurement. While its absolute value is not informative, its variance in a small neighborhood is a measure of the local roughness of each surface, which can help in discriminating smooth objects (*e.g.*, roofs). Since the variance of measured range increases with distance, we use the raw variance as well as the variance to mean ratio, also known as *dispersion index*.

In most cases, our features describe local properties, such as shape and orientation in a single-voxel neighborhood. However, other features depend on a larger context; for example, many of the height features depend on all the voxels above and below each voxel. Likewise, the features depending on rays depend on the whole path of the lidar rays that traverse each voxel.

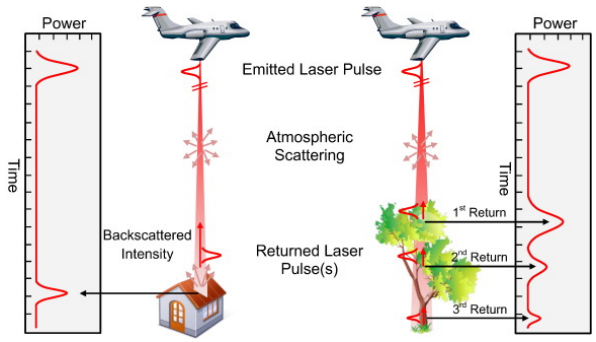
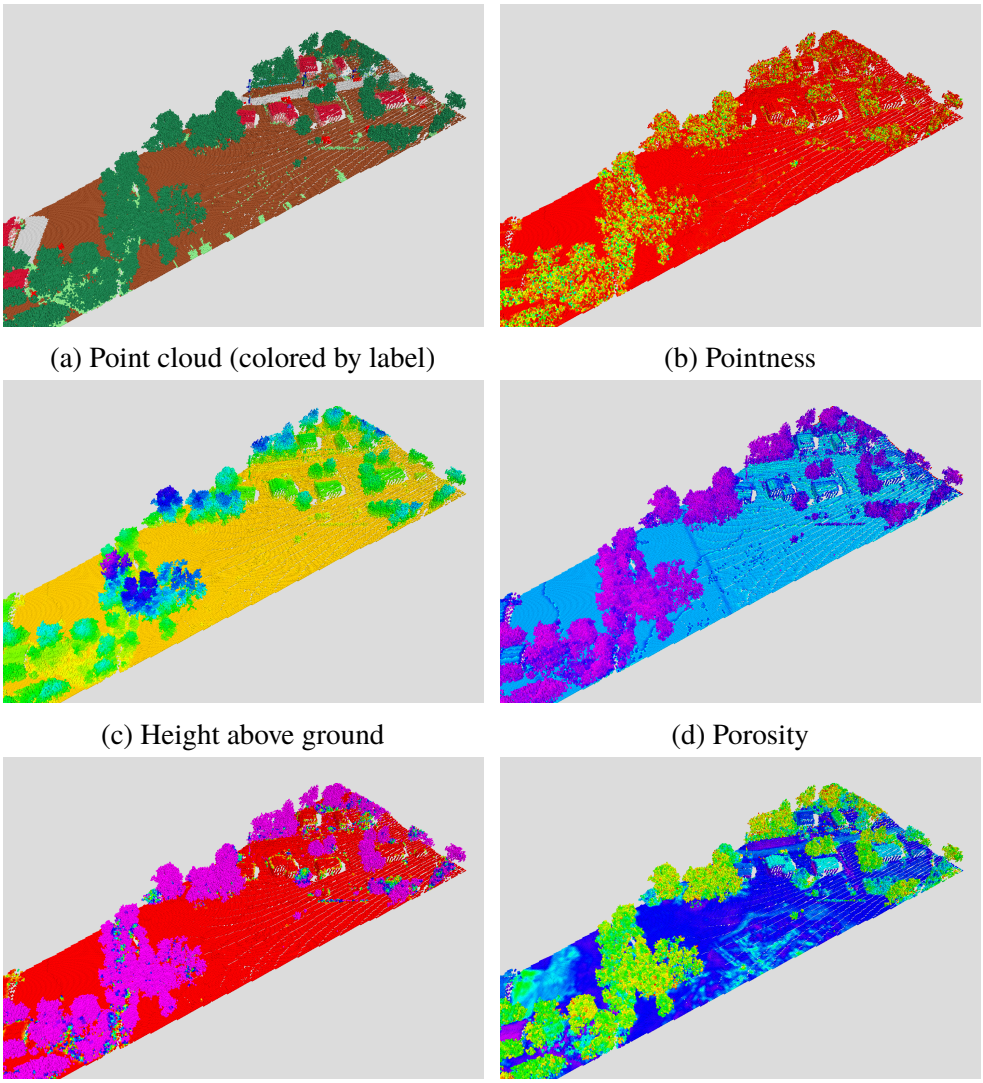


Figure 3.7: Illustration of multiple lidar returns in solid and porous objects. Figure reproduced with permission from [186].

Table 3.1: Point cloud features used in our system. See main text for more details.

Feature	Description	Citation
Local shape		
`pointness`	Smallest eigenvalue of local PCA	[103, 93]
`lineness`	Difference of largest and middle eigenvalue of local PCA	[103, 93]
`surfaceness`	Difference of middle and smallest eigenvalue of local PCA	[103, 93]
`vert-normal`	Dot product of surface normal to vertical vector	[156]
`vert-tangent`	Dot product of surface tangent to vertical vector	
`horiz-normal`	Dot product of surface normal with horizontal vector	
`horiz-tangent`	Dot product of surface tangent with horizontal vector	
Height and rays		
`points-above`	Number of points in voxels above	
`points-above`	Number of points in voxels below	
`rays-above`	Number of rays passing above this voxel	
`rays-below`	Number of rays passing below this voxel	
`porosity`	Ratio of ray hits to ray hits plus ray pass-throughs	[90]
`dist-above-min-0`	Height w.r.t. to lowest point below voxel	[156]
`dist-above-min-1`	Height w.r.t. to lowest point below voxel within radius of one voxel	[156]
`dist-above-min-2`	Height w.r.t. to lowest point below voxel within radius of two voxels	[156]
`dist-below-max`	Distance to highest point above voxel	
`density-ratio`	Ratio of points within this voxel relative to points above and below	
Lidar waveform		
`reflectance-mean`	Average reflectance of lidar pulses	
`reflectance-std`	Standard deviation of reflectance of lidar pulses	
`ret2-count`	Number of pulses which are interior returns	
`ret3-count`	Number of pulses which are final returns	
`ret2-ret3-ratio`	Ratio of interior to final returns	[29]
Range		
`range-std`	Standard deviation of measured range from lidar	[116]
`range-var-mean`	Ratio of lidar range variance to range mean (dispersion)	



(a) Point cloud (colored by label)

(b) Pointness

(c) Height above ground

(d) Porosity

(e) Interior to final return ratio

(f) Reflectance

Figure 3.8: Visualization of selected point cloud features used in this chapter. For each feature we display its per-voxel scalar value with a false color map in order to highlight its variation across the point cloud, rather than convey its absolute value.

Per-voxel classification

Figure 3.8 shows a point cloud from our dataset, along with false color representations of various per-voxel features for this point cloud. Qualitatively, it can be seen that different features are spatially correlated with different semantic labels. For example, *porosity* is high for vegetation, and low for buildings and ground; *height above ground* is high for both vegetation and objects above the ground, while low for the ground surface. This suggests that in combination, these features can be used to predict semantic labels.

As discussed in section 2.2, several machine learning methods exist to learn models that perform semantic label classification from labelled examples. Here, we frame the problem as simple supervised learning, in which we predict one of K labels for each voxel from its features.

We note that unlike several approaches in segmentation, we do not explicitly model the structured nature of the problem; that is, we treat each voxel as an independent instance in our dataset. This means our approach can not explicitly identify correlations between classes, *e.g.*, that building roofs are usually adjacent to building walls. This choice was made based on the observation that when using a rich feature set with a large spatial context, the trade-off in gained accuracy with respect to the added computational requirements was not favorable. However, recent approaches for efficient inference in structured models (*e.g.*, [83]) make them more attractive for use in future work.

The classifier we use for our system is a *decision forest*, a type of *random forest* [14], which consists of an ensemble of independently trained *decision trees*. See Criminisi, Shotton, and Konukoglu [39] for a survey on decision forests. This classifier offers several benefits: inference is computationally efficient, it can handle high-dimensional, possibly noisy and redundant features, and it is robust to variations in hyperparameters such as tree depth and number of trees in the ensemble. These advantages have made it a popular method in computer vision for 2D and 3D data, and it is used in other semantic point cloud labeling pipelines, *e.g.*, [24, 29, 159].

3.4 Experiments

3.4.1 Datasets

We found no publicly available datasets of lidar — either labeled or unlabeled — data with characteristics similar to those captured by our platform during low altitude ($20\,\mathrm{m}$ to $90\,\mathrm{m}$) flight. Existing public repositories of aerial lidar surveys, such as those collected by the USGS 3DEP [166] are geared towards applications in earth sciences and GIS. As such, they cover large areas, but are captured from a high altitude and have low point density (less than 10 points per m^2) relative to those of our platform (typically more than 50 per m^2). Moreover, they are usually unlabeled and are stripped of useful metadata, such as the lidar reflectance and sensor poses. On the other hand, lidar datasets captured at street-level, such as the CMU-Oakland dataset [126], have high point cloud densities but intrinsically have very different viewpoints and occlusion characteristics than the data in our application.

Therefore, we created a dataset using lidar data captured by our platform in fifteen test flights around western Pennsylvania throughout late 2014 and early 2015. We label nine semantic classes, described in Table 3.2. Figure 3.9 shows an example scene with points from all the classes.

Table 3.2: Per-voxel semantic class statistics of our dataset. Each voxel has dimensions $0.25 \times 0.25 \times 0.25\,\mathrm{m}^3$.

Class	Description	# Voxels	% Voxels
Ground	Rough ground (bare earth)	81182	25.88
Roof	Building roofs	6109	1.94
Wall	Building walls	1935	0.61
Low Veg.	Low vegetation (grass, shrubs)	32017	10.20
Canopy	Tree canopies	176838	56.38
Obstacle	Misc. objects (car, person)	1197	0.38
Pole	Utility poles	150	0.04
Wire	Power line	841	0.26
Paved	Smooth/paved ground	13338	4.25
Total		313607	100.0

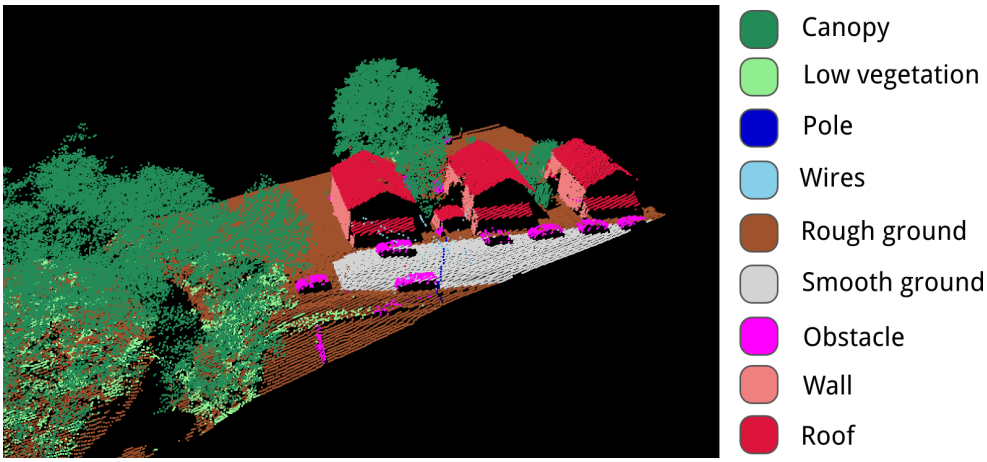


Figure 3.9: Example scene from our labeled point cloud dataset.

3.4.2 Results

Performance evaluation

To evaluate the performance of our system and the impact of using ground-surface aware features, we use per-class Precision, Recall and F1-scores across all voxels in the test set.

Table 3.3 summarizes the results. We can see that incorporating ground surface-aware features improves the evaluation metrics for almost every class, specially in regards to recall for *paved* and *obstacle*. For *paved*, this is explained by the fact that without the inferred ground surface, building roofs tend to appear similar to paved surfaces. For *obstacle*, this occurs because without the inferred ground surface, the height of objects on the ground is often estimated incorrectly. Both of these scenarios are illustrated in Figure 3.4.

Table 3.3: Precision (P), Recall (R) and F1-score for ((a)) baseline using naive height features and ((b)) our full system, replacing naive height features with ground surface-aware features. In ((b)), we highlight the changes (Δ) for each metric.

(a) Baseline

Class	P	R	F1
Ground	0.87	0.95	0.91
Roof	0.94	0.86	0.90
Wall	0.88	0.69	0.77
Low Veg.	0.78	0.52	0.62
Canopy	0.97	0.99	0.98
Obstacle	0.85	0.26	0.40
Pole	1.00	0.01	0.03
Wire	0.95	0.41	0.58
Paved	0.94	0.51	0.66
Avg.	0.90	0.58	0.65

(b) With ground surface-aware features

Class	P	Δ	R	Δ	F1	Δ
Ground	0.88	0.01	0.96	0.01	0.92	0.01
Roof	0.97	0.03	0.95	0.06	0.96	0.09
Wall	0.86	−0.02	0.75	0.03	0.80	0.06
Low Veg.	0.82	0.04	0.65	0.11	0.73	0.13
Canopy	0.98	0.01	0.99	0.0	0.98	0.0
Obstacle	0.85	0.0	0.50	0.23	0.63	0.24
Pole	0.92	−0.08	0.08	0.12	0.15	0.07
Wire	0.94	−0.01	0.67	0.2	0.78	0.26
Paved	0.96	0.02	0.87	0.25	0.91	0.36
Avg.	0.90	0.0	0.71	0.13	0.76	0.11

To gain further insight on the performance of our full system, we inspect the confusion matrix in Figure 3.10. We see that *poles* and *wires* are easily confused with *canopy*. This occurs because with our current set of features, these classes appear similar to each other; they tend to have high porosity, and occur at large heights. In addition, *pole* and *wire* are infrequent classes compared to *canopy*, so the classifier is biased to the latter. Other frequently confused classes are *obstacle* and *low vegetation*, which are often misclassified as *ground*. In both cases, this occurs because it is difficult to discriminate small objects and very low vegetation from each other and rough ground surfaces, even for humans.

Feature importance

Given that our feature set plays a critical role in the performance of our system, we are interested in which features are the most discriminative. Decision forests provide a coarse way of estimating the importance of each feature, the Gini importance metric [14]. This metric is derived from the decision tree induction process and quantifies, on average, how much each feature contributes to the classification output. Note that the metric provides a relative importance; its absolute value

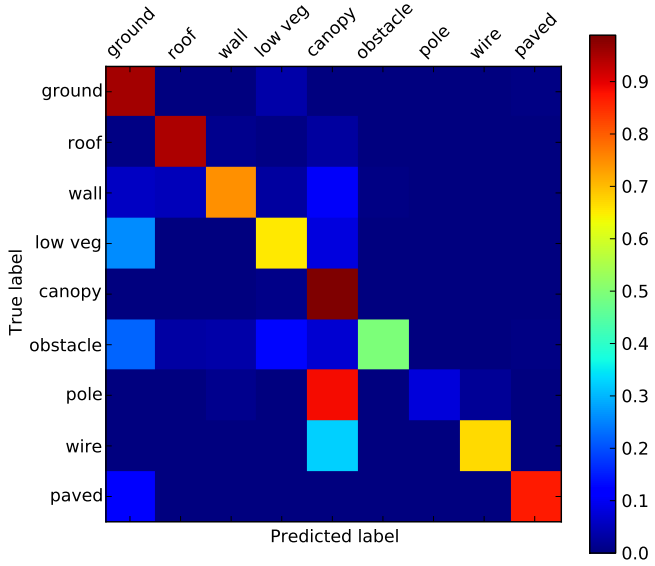


Figure 3.10: Confusion matrix of classifications in the test set

is not meaningful. According to this metric (plotted in Figure 3.11), the ratio of interior returns to final returns is the most important feature. Inspection of the data suggests this is due to its usefulness in discriminating tree canopies — which constitutes a large fraction of the dataset — from other classes. Porosity is also considered a highly important feature, for similar reasons to the return ratio. Finally, the third most important metric is the *density ratio*, which compares density of points relative to each column, and helps for shapes with different vertical distributions.

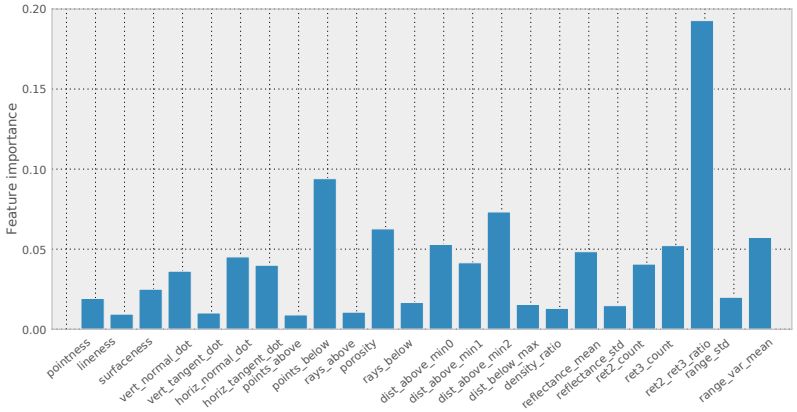


Figure 3.11: Point cloud feature importance ranking, obtained with the Gini importance metric

Timing

The system must be able to update with low latency in order to be useful for planning purposes. For our application, the latency target from point cloud reception to semantic map update was set to be up to $2\,\mathrm{s}$.

We timed our pipeline on a system with a $2.9\,\mathrm{GHz}$ Intel i3 CPU with $8\,\mathrm{GB}$ of RAM. For a semantic map of $100 \times 100 \times 100\,\mathrm{m}^3$ (*i.e.*, $400 \times 400 \times 400$ voxels) and an average input of $20\,000$ points per s, we have an average latency of $1.4\,\mathrm{s}$. Table 3.4 shows how this timing breaks down in terms of various steps in our pipeline. The most onerous operation in our system is spent updating the sufficient statistics for each grid cell. We found that in this step, raytracing is the bottleneck, occupying around 80% of it. This is due to the fact that each ray may traverse a large number of voxels. The next main bottleneck is classification, which in this table includes two passes — one for the ground inference, and another for the final predicted labels. Other steps have a relatively small contribution.

Our system is not heavily optimized and could be significantly sped up with some minor modifications. Currently, none of our steps take advantage of multithreading. However, many of them are easily parallelized. For example, classification and feature extraction could be parallelized over subregions of the grid map (or even on a per-voxel basis), and likewise for sufficient statistics updates, other than raytracing.

Table 3.4: Average latency in ms per $20\,000$ points

Step	Time (ms)
Sufficient statistics	838.9
Classification	369.6
Feature extraction	72.8
Ground interpolation	23.3
Scrolling	6.0
Others	4.1
Total	1314.6

Qualitative segmentation results

We show several screen captures of our approach operating in real time in Figure 3.12. The results are generally plausible, particularly for the more common classes. However, we in the last panel, we can see two error modes. *Ground* (brown) is confused with *paved* (gray); this is a relatively common issue, given the similarity of the two classes, even to human labelers.

Another visible (and potentially serious) error are the points labelled as *roof* (red), which in reality correspond to *ground* or *low vegetation*. This is attributable to the mound-like elevation of this surface relative to the surrounding area, which is unusual in the dataset.

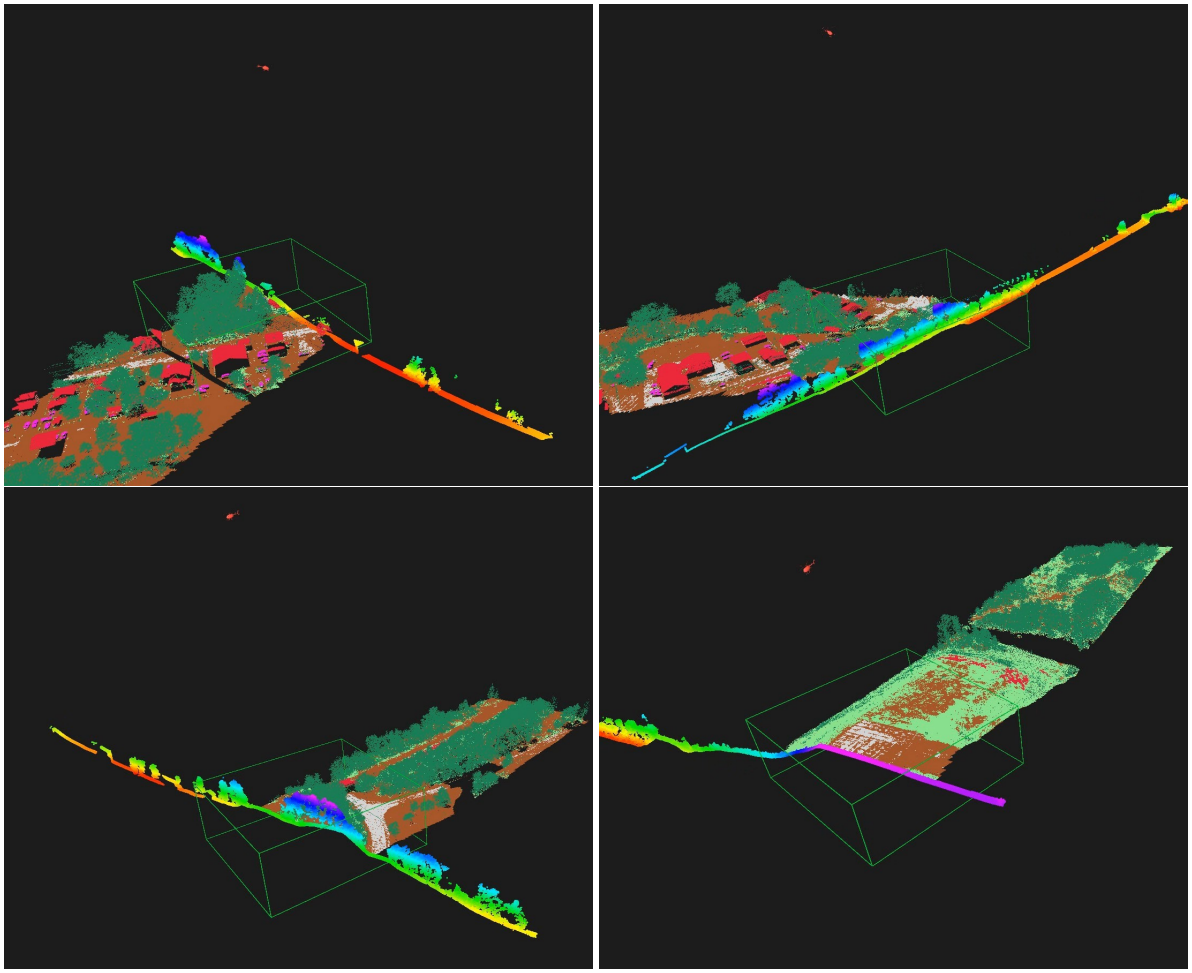


Figure 3.12: Screen captures of our system performing labeling in real time. The pose of the helicopter is shown with a red CAD model (zoom may be required). The green wireframe box is the ROI used in segmentation at each time. Segmented point clouds outside of the ROI are retained and visualized within a finite FIFO queue. Video is available at https://youtu.be/wuv0IyKkZmY.

Interpolation for active Landing Zone evaluation

One of the capabilities that our team developed for the autonomous helicopter is active sensing for possible landing zones [4]. Due to to the relative sparsity of lidar point clouds at long distances, deliberate control of the lidar sensor to acquire scans of potential landing zones greatly increases the ability of the helicopter to perform safe landings after a high-speed approach. Our team realized that a good estimate of the height of the ground surface would increase the accuracy and speed of this search, which led us to adapt the ground surface interpolation module for this purpose. While the module was not evaluated quantitatively, in our testing it performed qualitatively well and was integrated in the final system for autonomous landing. Figure 3.13 shows captures of this system incrementally estimating the ground surface during a landing approach.

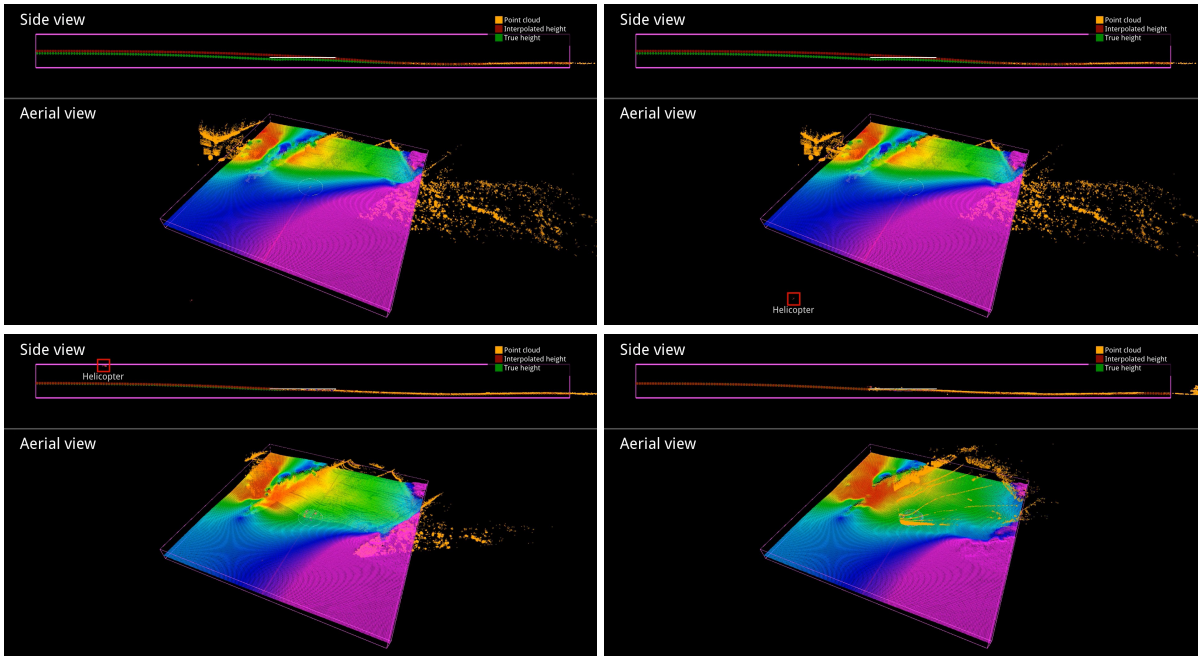


Figure 3.13: Ground surface interpolation for active sensing of landing zones. In each panel, we show on the top a profile view of the estimated ground surface and the true ground surface. On the bottom, we show an oblique view of the surface, false-colored by height. The helicopter is also shown, and it is highlighted in the third and fourth panels. Video is available at https://youtu.be/wNrkhvdUbfo.

3.4.3 Discussion and limitations

Qualitatively, our system performs with low enough latency and sufficiently high accuracy to aid navigation in several ways, given its high performance in classifying building walls and roofs, ground, and tree canopies. In addition, we have shown that our novel ground surface inference provides a significant improvement in the detection of these and other classes.

However, some classes, such as *obstacle*, *wire* and *pole*, are challenging for our approach. The relatively poor performance in these classes is particularly concerning given the danger they pose

to aerial vehicles.

We believe that a large contributing factor for the poor performance is the relative rarity of these classes in the dataset. The relatively low number of examples for these classes means there are fewer examples to generalize from. Moreover, as our current training objective function aims to minimize the average accuracy over all classes and examples, it will tend to pay less attention to infrequent classes, and will label ambiguous cases with the more common label.

One natural strategy to address this issue is to collect and label more data for the challenging cases. In lieu of additional data, techniques such as modified training objectives and data augmentation could mitigate this issue.

Another direction that would potentially improve our performance is modeling spatial and semantic context in a more expressive way. Currently, we rely on our features to encode various types of spatial context, but it is still relatively limited, and more features encoding information at various levels of spatial resolutions could be beneficial. However, it is difficult to design these features by hand, as it involves a fair amount of guesswork and inspection of the data so as to encode relevant information. In chapter 4 we present an alternative to learn features along with the classifier.

Explicitly modeling semantic context would also be beneficial, as it could allow the model to learn, *e.g.*, that wire voxels are near pole voxels, and also near other wire voxels. For this we could use graphical models [125, 158] or cascaded classification [185].

Finally, other sensing modalities would also be useful. For example, images have a higher resolution than lidar, which would allow better discrimination of small objects such as wires, as well as provide color and texture information that may be beneficial for a variety of classes. While not discussed in this thesis, we developed an image-based method for wire detection in Madaan, Maturana, and Scherer [117]. We discuss multimodal architectures incorporating image information in chapter 5.

3.5 Summary

We have presented a semantic mapping system for aerial LIDAR that can successfully create semantic maps in real time from streaming point cloud data. Our system features a novel ground surface inference scheme that significantly boosts classification accuracy of various classes.

However, as we discuss in section 3.4, while the accuracy of our system is acceptable for many classes of interest, we have found that it struggles with others. We believe the main cause is that our current set of hand-engineered features does not encode sufficient information to allow these classes to be detected robustly by the classifier. While we can always design more features, this is a laborious task. In the next chapter, we instead explore a method that can learn discriminative features for 3D point cloud data.

Chapter 4

Deep learning for efficient and robust point cloud classification

In chapter 3, we presented a system to create semantic maps from 3D point clouds for an autonomous helicopter. We found that this system struggles to classify certain classes relevant for this application, and we hypothesized that this was largely due to the limited amount of information in the hand-engineered point cloud features at the heart of the system. Rather than design or further tune these features, a laborious process, in this chapter we explore the use of deep neural networks to jointly learn the features and classifiers.

To this end, we present VoxNet, a novel architecture for fast and accurate classification of point cloud data with 3D Convolutional Neural Networks (CNNs). The key innovation of this architecture is to the application of modern convolutional architectures – which so far have been mainly used for image and audio data – to the processing of 3D point cloud data, by applying 3D CNNs to a volumetric occupancy map. We show the benefits of this architecture in a Landing Zone (LZ) detection task for our autonomous helicopter, as well as more generic object classification tasks with lidar point clouds in an urban self-driving scenario, RGBD point clouds of indoor objects, and CAD models.

4.1 A deeper look at point cloud data

The needs that motivated the work in this chapter are similar to those of chapter 3. As before, we are interested in improving and expanding the capabilities of autonomous rotorcraft operating in unstructured environments. However, here our focus is different. In chapter 3, we built a semantic mapping system for a relatively broad set of categories relevant for flights in urban and rural locations: *tree canopy*, *building wall*, *ground*, *obstacle*, etc. While the system performs well for most classes, it struggles for others (*e.g.*, detecting small obstacles) that are particularly relevant for one of our main tasks of interest, autonomous *landing zone* (LZ) detection.

Helicopters have the advantage of being able to land on unprepared sites. For autonomous unmanned helicopters to fully exploit this ability, they must be capable of accurately and reliably assessing the safety of potential landing sites. The task of LZ detection is to detect zones within a predefined area that are safe for the helicopter to land on, which are not necessarily marked

or otherwise prepared [81]. Approaches using simple geometric point cloud analysis have been demonstrated ([87, 183, 153]), but these are incapable of correctly detecting LZs in cluttered locations, such as the one shown in in Figure 4.1. This is because the geometry of soft vegetation, which is safe to land on, is similar to the geometry of obstacles which are not safe to land on. Moreover, the system in chapter 3, despite being equipped with a far more comprehensive set of point cloud features and a more sophisticated classifier, also struggles with this kind of scenario.

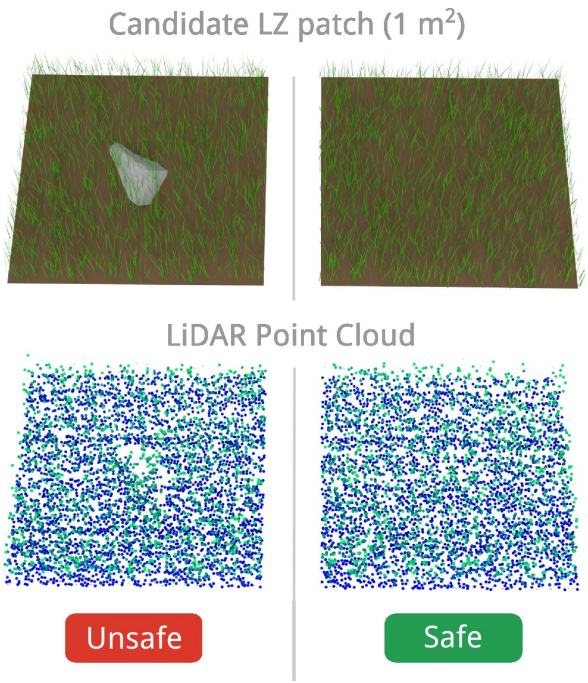


Figure 4.1: Point clouds for two candidate landing zones, one of them unsafe

As in our last chapter, we also seek a solution capable of running in real time, which may be particularly pressing for this application, as the output of the system becomes most useful in the "endgame" of the landing process, when it must commit to landing or abort as it nears a candidate LZ.

To address the challenges posed by this task we developed a solution leveraging recent advances in deep neural networks, that can learn features and classifiers jointly from data.

As we developed our approach, we hypothesized that our framework, which is capable of learning *ad hoc* features for each task, rather than relying on a pre-built set of features, could also prove useful for more generic tasks in classification with 3D data. Thus, we also explore the application of VoxNet to the classification of objects from lidar for self-driving applications [176, 172], classification of objects captured from indoors RGBD data [129], and classification of CAD models [184].

4.1.1 Problem statement

If we abstract various details surrounding the implementation of our system in real scenarios, we can simply frame the problem we address in LZ detection and object recognition as 3D point cloud

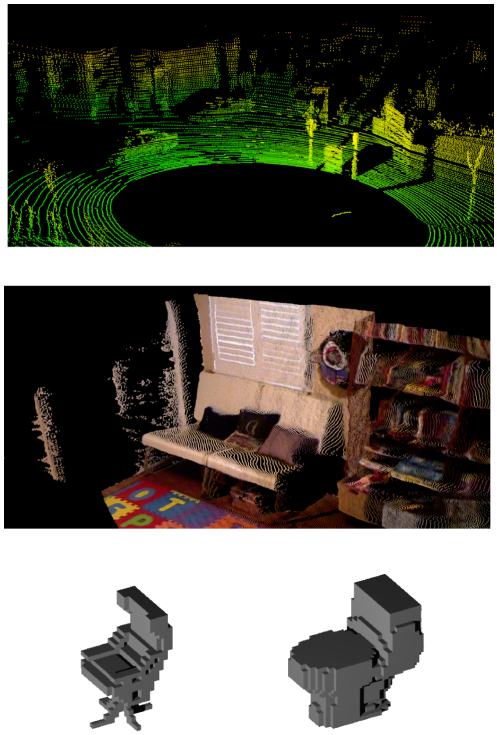


Figure 4.2: For our object recognition experiments, we use data from lidar [49] (top), RGBD sensors [129] (middle), and voxelized CAD models [184] (bottom).

classification:

Given a 3D point cloud, classify it as one of K predefined classes.

Concretely, for the LZ detection task, we pose the problem as binary classification by classifying small areas as *safe* or *unsafe*; for the object recognition tasks, we classify each input as one of the classes defined by its corresponding dataset, *e.g.*, one of *car*, *truck*, *pedestrian*, etc. for the urban driving dataset, or *table*, *chair*, *sofa*, etc. for the indoors dataset.

In most of this chapter, we will focus on this task in isolation. However, in practice, solving this problem by itself is insufficient for a semantic mapping system, which is our original goal. We will describe how we use VoxNet for the task of semantic mapping of LZs.

4.2 Related work

Landing Zone detection systems An early approach using simulated lidar for landing zone detection is Johnson et al. [87]. They propose a system for landing zone selection based on a relatively simple geometric analysis of terrain roughness and slope. Whalley et al. [183] and Scherer et al. [153] use similar geometric criteria and demonstrate its success in real data. However, as we show in our experiments these approaches fail in terrain cluttered with vegetation.

Ground filtering and terrain modeling Ground filtering and terrain modeling methods, as those described in section 3.2, could also be considered relevant for LZ detection, specially in the case of terrain cluttered with vegetation. However, as mentioned before, they make several assumptions that do not apply to the scenarios encountered by a vehicle flying very close to the ground. Moreover, they are generally designed to discriminate ground points from non-ground points; however, this does not translate directly to an LZ safety assessment, specially in the presence of obstacles such as rocks.

Traversable vegetation and obstacle mapping Another relevant line of work proposes to discriminate traversable grass from hard obstacles using heuristics based on the local statistics of laser scan lines ([116, 119]). These systems assume measurements are taken from forward-facing scanners at very close range on ground vehicles. We have observed that the features used by this method, which we use in chapter 3 and experiments in this chapter, are not highly discriminative, specially with sparse point clouds. Moreover, these methods do not distinguish between the ground and other solid surfaces, and would need to be modified on this account.

Semantic classification with lidar As outlined in section 3.2, there are several methods to classify and segment point clouds with lidar. These methods could be applied to LZ detection, *e.g.*, by segmenting *ground*, *grass* and *obstacle* points or voxels. However, this is still does not necessarily translate into an assessment of safety for a larger area, and another layer would have to be engineered to make this prediction. This is nontrivial, specially considering the effects of sparsity and occlusion. Regardless, we have taken various hand-engineered point cloud features from work in this area to build one of our baselines.

Another issue with prior work in this area, that affects LZ detection as well as generic object recognition, is the reliance on hand-engineered features that may not encode the information necessary for each task. Instead, our model learns to extract discriminative features directly from a voxelized representation.

Learning features for 3D data As we describe in chapter 2, in computer vision there has been a trend towards learning features from data instead of (or in addition to) using hand-engineered features, even preceding the popularity of deep learning. However, following the success of deep learning, and particularly, Convolutional Neural Networks (CNNs) in various benchmarks, they have become the dominant framework for feature and classifier learning.

The trend towards feature learning also occurs in the analysis of 3D data, but so far there has not been work to learn features with CNNs in volumetric data.

Several authors have extended CNNs to use RGBD data ([107, 163, 155]). These approaches leverage existing CNN architectures for 2D data by treating the depth channel as an additional channel alongside the RGB channels.

Gupta et al. [74] propose a similar idea, but instead of raw depth they use channels encoding pixelwise height and surface orientation from the depth data. While straightforward, these methods do not not make full use of the geometric information in the data and make it difficult to integrate information across viewpoints.

For lidar, Quadros et al. [142] propose a feature that describes scans locally with a 2.5D representation, and [49] studies this approach in combination with a form of unsupervised feature learning. Like the RGBD methods, these methods are still 2D-centric. Our work differs from these in that we employ a fully volumetric representation, resulting in a richer and more discriminative representation of the environment.

There has also been work in feature learning for volumetric data. Flitton et al. [60] apply a biologically inspired neural network to the classification of computed tomography (CT) imagery. While their architecture, at a high level, is similar to ours, they do not learn features from the raw data, but instead design the network with hard-coded feature extractors at various levels.

3D CNNs have been applied to other domains. Viewing video data as a volume with time as the third dimension, Ji et al. [86] and Karpathy et al. [88] apply 3D CNNs to human action classification. In terms of their components and operations, these networks work in the same way as ours, but the nature of the data is very different.

Recently, [101] proposed an unsupervised volumetric feature learning approach as part of a pipeline to detect indoor objects from RGBD data. This approach is based on sparse coding, which is generally slower than CNNs.

Concurrently with our initial publication on VoxNet [120], Wu et al. [184] proposed a generative 3D convolutional model of shape and applied it to recognition of CAD objects, among other tasks. This method is close to ours, but they use a significantly larger network that is trained as a generative, rather than discriminative, network. We compare this approach to ours in the experiments.

4.3 Approach

Given a point cloud, our task is to predict a class label. Our system for this task has two main components: a volumetric grid representing our estimate of spatial occupancy, and a 3D CNN that predicts a class label directly from the occupancy grid. The process is depicted in Figure 4.3. We describe each component below.

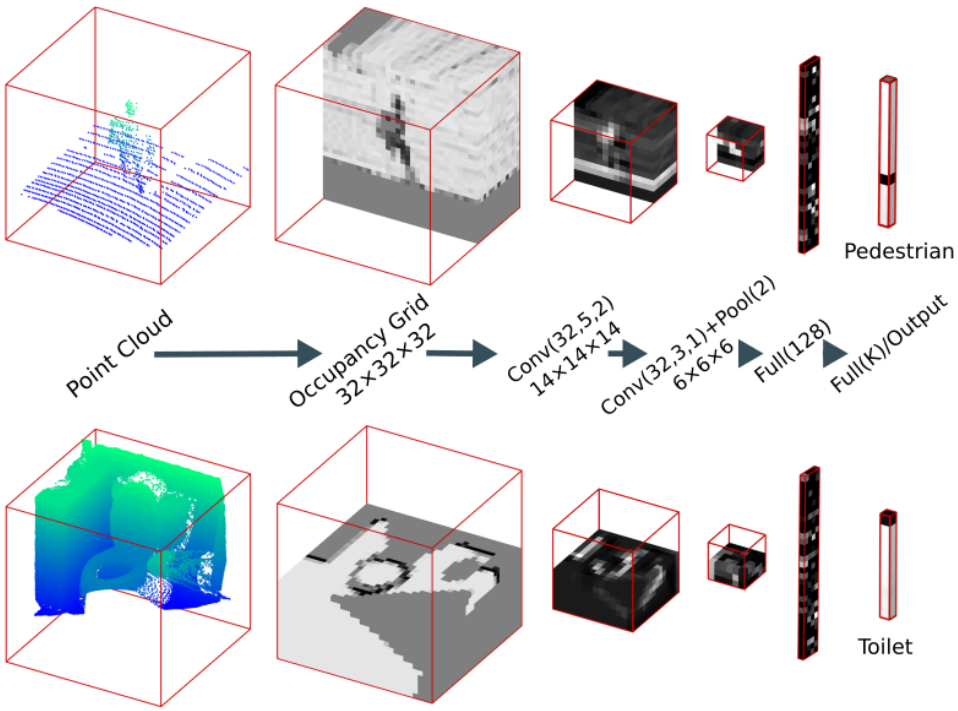


Figure 4.3: VoxNet architecture and data flow for two example inputs. The system receives 3D point clouds, which are voxelized as occupancy grids and then passed through a 3D CNN to predict a label.

4.3.1 Volumetric occupancy grids

Occupancy grids [122, 169] represent the state of the environment as a 3D lattice of random variables (each corresponding to a voxel) and maintain a probabilistic estimate of their occupancy as a function of incoming sensor data and prior knowledge.

There are two main reasons we use occupancy grids. First, they allow us to efficiently estimate *free*, *occupied* and *unknown* space from range measurements, from measurements coming from potentially different viewpoints and time instants. This representation is richer than those which only consider occupied space versus free space such as point clouds, as the distinction between free and unknown space can potentially be a valuable shape cue, as suggested in the example shown in Figure 4.4. Here the shape of the car is more visible when explicitly distinguishing free space from unknown space (which in this case correspond to cells occluded by the car's surface).

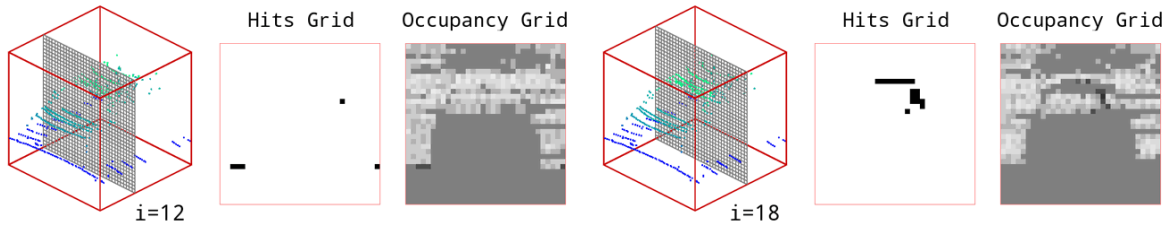


Figure 4.4: Two cross-sections comparing occupancy representations. We show a point cloud of a car and the section, indexed by i. We then compare a grid representing only occupied space (dark pixels), with a grid representing occupied space (darker pixels), free space (lighter pixels) and unknown (gray pixels).

Second, they can be stored and manipulated with simple and efficient data structures. In this work, we use dense arrays to perform all our CNN processing, as we use small volumes ($32 \times 32 \times 32$ voxels), and GPUs are well suited for processing dense data. To process larger spatial extents, as in the task of LZ detection, we store all volumetric data we cover in CPU memory and copy small volumes to the GPU memory as needed. To manage this data, we use OpenVDB [128], a semi-sparse data structure that stores volumetric data in dense "tiles" organized in a shallow hierarchy.

Reference frame and resolution

In our volumetric representation, each point (x, y, z) is mapped to discrete voxel coordinates (i, j, k) in our ROI, similarly to our system in chapter 3. The mapping is a uniform discretization but depends on the origin, orientation and scale of the voxel grid in space. The appearance of the voxelized surface depends heavily on these parameters, similarly to how the appearance of an object in a two-dimensional image depends on the position, orientation, and resolution of the camera sensor.

For the origin of our ROI, we assume it is given as an input, *e.g.*, obtained by a segmentation algorithm or given by a sliding box. To make our method more robust to noise in the ROI origin, we augment the training data with random translations of the point cloud.

For the orientation of the ROI, we align its Z axis with the negative of gravity, or in the case of CAD models, with an "up" direction that serves a similar purpose. In our datasets this information is available. However, this only constrains two degrees of freedom. The yaw, or rotation around the Z axis, could be determined in different ways, depending on the application. In a robotic context, it could be set according to an inertial frame or relative to an egocentric frame, but for CAD models neither of these apply. For simplicity, we opt to make our CNN robust to the yaw of the input by augmenting the training data with multiple rotations per point cloud. Test time augmentation was also applied for the object recognition tasks.

For the voxelization scale, we adopt two strategies, depending on whether the input data has a known scale relative to a fixed reference, such as physical distance units, or whether its scale is arbitrary or unknown.

The first case applies to the lidar and RGBD datasets we use in this chapter (as well as chapter 3),

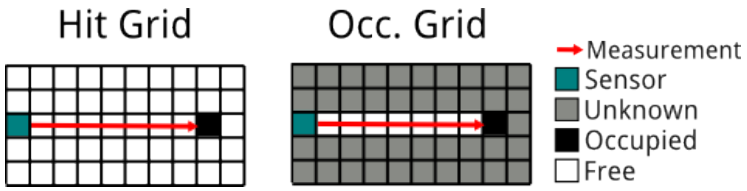


Figure 4.5: Occupancy models. In the Hit grid model (left), each cell is modelled as occupied (black) or not known to be occupied (white). No distinction is made on free space versus unknown space. In the binary occupancy grid, the belief for each cell is a continuous variable ranging from 0 to 1, indicating whether it is known to be occupied (black), free (white) or unkown (gray). In the density grid, the intermediate values have a slightly different interpretation than "unknown"; see the main text.

where the data has physical units of distance. In this case, the size of our voxels is defined relative to this unit (*e.g.*, $0.1 \times 0.1 \times 0.1\,\mathrm{m}^3$), and assumed to be constant for any given experiment. We evaluate different voxel resolutions in the experiments.

We encounter the second case in the 3D CAD dataset we use in this chapter [184], which uses models from heterogeneous sources that do not necessarily follow any given convention for their measurement units. In this case, we follow the strategy adopted by the authors of the dataset. We isotropically rescale the geometry of each model to fit in an arbitrary canonical volume (*e.g.*, a unit cube), and define the size of each voxel (again, fixed for each experiment) in the same units as this volume. In this case, the units have no consistent relation to physical space.

Occupancy models

There are many possible ways of encoding point clouds obtained from range measurements into a volumetric grid. Here we consider three models that model *occupancy*, *i.e.*, whether a grid cell is occupied. We refer to these as *Binary Occupancy Grid*, *Density Grid* and *Hit Grid*.

Let $\{z^t\}_{t=1}^T$ be a sequence of range measurements that either hit ($z^t = 1$) or pass through ($z^t = 0$) a given voxel with coordinates (i, j, k). Assuming an ideal beam sensor model, we use 3D ray tracing [1] to calculate the number of hits and pass-throughs for each voxel.

Binary occupancy grid: Introduced by Moravec and Elfes [122], this is the *de facto* standard model in robotics, and the term *Occupancy grid* is often used synonymously with this model. In this model, each voxel is assumed to have a binary state, occupied or unoccupied. The probabilistic estimate of occupancy for each voxel is computed with log odds for numerical stability. Using the formulation from Thrun [169], we update each voxel traversed by the beam as

$$l_{ijk}^t = l_{ijk}^{t-1} + z^t l_{\text{occ}} + (1 - z^t) l_{\text{free}} \tag{4.1}$$

where l_{occ} and l_{free} are the log odds of the cell being occupied or free given that the measurement hit or missed the cell, respectively. We set these to the values suggested in Hähnel, Schulz, and Burgard [77], $l_{free} = -1.38$ and $l_{occ} = 1.38$, and clamp the log odds to $(-4, 4)$ to avoid numerical issues. Empirically, we found that within reasonable ranges these param-

eters have little effect on the final outcome. The initial probability of occupancy is set to 0.5, *i.e.*, $l^0_{ijk} = 0$. In this case, the input to the network are the log-odd values l_{ijk}.

Density Grid: This model is similar to the standard occupancy grid, but allows for a different interpretation. In this model, each voxel is assumed to have a continuous "density"[1], corresponding to the probability the voxel would block a sensor beam. We use the formulation from Tipaldi, Spinello, and Burgard [173], where we track the Beta parameters α^t_{ijk} and β^t_{ijk}, with a uniform prior $\alpha^0_{ijk} = \beta^0_{ijk} = 1$ for all (i, j, k). The update for each voxel affected by the measurement z_t is

$$\alpha^t_{ijk} = \alpha^{t-1}_{ijk} + z^t$$
$$\beta^t_{ijk} = \beta^{t-1}_{ijk} + (1 - z^t)$$

and the posterior mean for the cell at (i, j, k) is

$$\mu^t_{ijk} = \frac{\alpha^t_{ijk}}{\alpha^t_{ijk} + \beta^t_{ijk}} \tag{4.2}$$

Kelly et al. [90] used a similar model (equivalent to setting $\beta_{ijk} = 0$) to model the density of each cell, without an explicit probabilistic interpretation.

In the density grid formulation, the value of μ^t reflects our best belief regarding the probability a lidar beam would traverse the voxel, rather than uncertainty originating from a lack of observations. This uncertainty could be expressed by the posterior variance of z^t, or directly via the conjugate posteriors of α^t and β^t, and added as additional channels in the occupancy grid. However, this multiplies the required amount of memory, and preliminary experiments in this direction did not show any improvement. Thus, in this case we use μ_{ijk} as input to the network.

Hit grid: This model only consider hits, and ignores the difference between unknown and free space. Each voxel has an initial value $h^0_{ijk} = 0$ and is updated as

$$h^t_{ijk} = \min(h^{t-1}_{ijk} + z^t, 1) \tag{4.3}$$

This model can be seen as a simple discretization of the point cloud. While this model discards some potentially valuable information, in our experiments, it performs surprisingly well. Moreover, it does not require raytracing, which is useful in computationally constrained situations. Another advantage is that it does not require knowledge of the origin of each ray; while this information is usually available when recording a dataset, it is often not provided with processed point clouds.

4.3.2 3D Convolutional Neural Network architectures

There are three main reasons 3D CNNs are an attractive option for our task.

First, they can potentially extract and integrate information from the occupancy data at multiple scales and levels of abstraction. For example, filters in the first layer can encode plane-like or

[1] In the physical sense, not in the sense of a probability density function.

corner-like structures at various orientations, and subsequent layers can construct a hierarchy of more complex features representing larger regions of space, eventually leading to a global label for the input occupancy grid.

Second, when trained with a differentiable loss function, their weights can be learned end-to-end in a supervised manner using Stochastic Gradient Descent (SGD). This elides the need for methods that separately learn or hand-engineer the "feature extraction" and "classifier" parts of the CNN, decreasing the engineering burden and increasing the likelihood that the learned features will be useful for the task at hand. Moreover, by using the negative log-likelihood as our loss function, the model's predictions can be interpreted as probabilities, which we can use to manage risk and guide sensing.

Finally, inference in CNNs has attractive properties for use in real-time robotics. Inference is purely feed-foward and uses a fixed number of operations for any given neural network architecture and input size, making its runtime highly predictable. Furthermore, most operations can be performed efficiently on commodity graphics hardware.

Below we describe the different layer types considered for our 3D CNN architecture. For each layer type, we use a shorthand description in the format $Name(hyperparameter)$.

Input Layer: This layer accepts a fixed-size grid of $I \times J \times K$ voxels. In this work, we use $I = J = K = 24$ or $I = J = K = 32$, depending on the dataset. While larger grids may be desirable to capture higher spatial resolution or larger spatial extents, the computational and storage resources required scale cubically with the number of voxels per dimension, making inference too slow (or altogether infeasible) with our current implementation and target platforms.

The input for each cell is updated according to the occupancy model, *e.g.*, Equation 4.2. In all cases we subtract 0.5 and multiply by 2, so the input is in the $(-1, 1)$ range; no further preprocessing is done. While this work only considers scalar-valued inputs, our implementation can trivially accept additional values per cell, such as lidar intensity values or RGB information from cameras.

Convolutional Layers $C(f, d, s)$: These layers accept four-dimensional input volumes in which three of the dimensions are spatial, and the fourth contains feature maps. The layer creates f feature maps by convolving the input with f learned filters of shape $d \times d \times d \times f'$, where d are the spatial dimensions and f' is the number of input feature maps. Convolution can also be applied at a spatial stride s. The output is passed through a leaky rectified nonlinearity unit (ReLU) [115] with parameter 0.1.

Pooling Layers $P(m)$: These layers downsample the input volume by a factor of by m along the spatial dimensions by replacing each $m \times m \times m$ non-overlapping block of voxels with their maximum.

Fully Connected Layer $FC(n)$: Fully connected layers have n output neurons. The output of each neuron is a learned linear combination of the outputs from the previous layer, passed through a nonlinearity. We use ReLUs save for the final output layer, where the number of outputs corresponds to the number of class labels and a softmax nonlinearity is used to provide a probabilistic output.

Given these layer types and their hyperparameters, there is an infinite number of architectures

that can map a volumetric grid to a vector of predictions. In the image domain, networks such as AlexNet [98] and VGG [161] have served as a useful template for a great deal of other architectures. While these architectures can be directly lifted to 3D, it is unclear whether this is the best choice, specially in light of the higher computational demands of volumetric networks.

To investigate this issue, in subsection 4.4.1 we perform an extensive stochastic search over hundreds of 3D CNNs architectures, evaluating the model performance on a synthetic LIDAR dataset.

Based on those experiments, in this chapter we use three models. The first two, used in the LZ detection experiments, are two high-performing models with different depths, which we denote VoxNet-LZ1 and VoxNet-LZ2 for brevity. They are described in subsection 4.4.1.

As we describe in subsection 4.4.3, for the object recognition experiments we adopt a modified version of VoxNet-LZ2, which preserves its depth but uses smaller filters and a reduced fully connected layer, making it faster and easier to learn. This model is illustrated Figure 4.3. We refer to this model simply as VoxNet[2].

4.4 Experiments

In our initial set of experiments we will describe the exploration of architectures that led to our initial version of VoxNet.

We will then describe the application of our architectures to different tasks. In the first, we classify terrain scanned by a lidar sensor as being safe or unsafe for a helicopter landing. In the second, we perform multiclass object recognition in three domains: CAD models, urban lidar point clouds, and indoor RGBD point clouds.

4.4.1 Exploration of architectures

Dataset

To evaluate different 3D CNNs we created a synthetic dataset intended to approximate the LZ detection task. The dataset consists of simulated 3D scans representing various patches of terrain with varying amounts of grass and different number of obstacles; the task of the CNNs is to discriminate whether a patch is *safe*, *i.e.*, has no obstacles. We use synthetic data for this task as it is a simple way to generate large amounts of diverse data with known ground truth.

We recreate the scanning pattern of our lidar sensor by simulating its pulse repetition rate, angular resolution, and the nodding behavior of its sensor mount. Our sensor is a RIEGL configured to scans lines perpendicular to the direction of flight with up to 100° horizontal field of view (FOV). It is mounted on a custom motorized platform for nodding, optionally allowing up to 100° vertical FOV scanning. The sensor assembly is mounted on an helicopter which uses an INS for global registration of the point cloud.

We also add Gaussian noise to the range, based on the manufacturer specifications ($\sigma = 25\,\mathrm{mm}$). Small motions for the origin of the sensor pose were added by a Gaussian random walk with zero

[2]The reason behind this nomenclature is that we first used the name VoxNet for this specific model. VoxNet-LZ1 and VoxNet-LZ2 are two closely related predecessors that were formerly nameless.

mean and $\sigma = 2\,\mathrm{cm/s}$. Scenes were scanned until a point density of $3000\,\mathrm{points/m^2}$ was reached, consistent with the density obtained near the landing zones in the autonomous landing missions from [35].

The generated lidar rays are then intersected against a synthetic scene consisting of a ground surface, grass blades, and box-shaped obstacles. The ground surface is a mesh in which the height of the vertices is perturbed by Perlin noise with a height range of $-0.05\,\mathrm{m}$ to $0.05\,\mathrm{m}$. The grass blades are simulated by a 3-triangle strip of maximum width $3\,\mathrm{mm}$, and normally distributed height and inclination. The grass placement is generated according to a homogeneous spatial Poisson process with a configurable intensity, as in [116]. The box is generated with dimensions $0.15 \times 0.15 \times 0.15\,\mathrm{m^3}$ dimensions at a uniformly random location and yaw angle on the plane.

Various parameters were systematically swept as shown in Table 4.1. We generate 20 instances per parameter setting, resulting in 26860 total instances (note that this does not consider any augmentation performed in training). We add obstacles to half the instances; these are considered *unsafe*, while the rest are considered *safe*.

Table 4.1: Simulation parameter sweep for synthetic datasets.

Parameter	Values
Blade per $\mathrm{m^2}$	$100, 200, \ldots, 1800$
Scanline angular resolution $^\circ$	0.05, 0.1, 0.2 and 0.41
Blade width mm	3, 5 and 8
Blade mean height m	0.1, 0.2 and 0.3
Sensor distance in x-axis m	-5.0 and -10.0
Sensor height m	4.0 and 8.0

Comparison of simulated and real data

As a sanity check of the relevance of our synthetic datasets we constructed a calibrated target and scanned in the laboratory. The target and a render of its synthetic version, along with a range histogram of 1500 points and an occupancy grid are depicted in Figure 4.6. Note that the blades are randomly generated and not meant to match the real grass on an individual blade level.

Evaluation metric

As described earlier, our task has the form of binary classification, with categories *safe* and *unsafe*. For our initial exploration, we use accuracy to evaluate each 3D CNN, computed as the fraction of instances where the hard classification prediction (computed with a fixed threshold of 0.5) matches the ground truth label. We use this metric as a simple way of ranking among models. In subsequent experiments (subsection 4.4.2) we use more fine-grained metrics.

CNN architecture and hyperparameters

To find suitable architectures for our 3D CNNs, we search over a parameterized space. We follow a general pattern followed by popular image-based CNNs, with alternating convolution and pooling

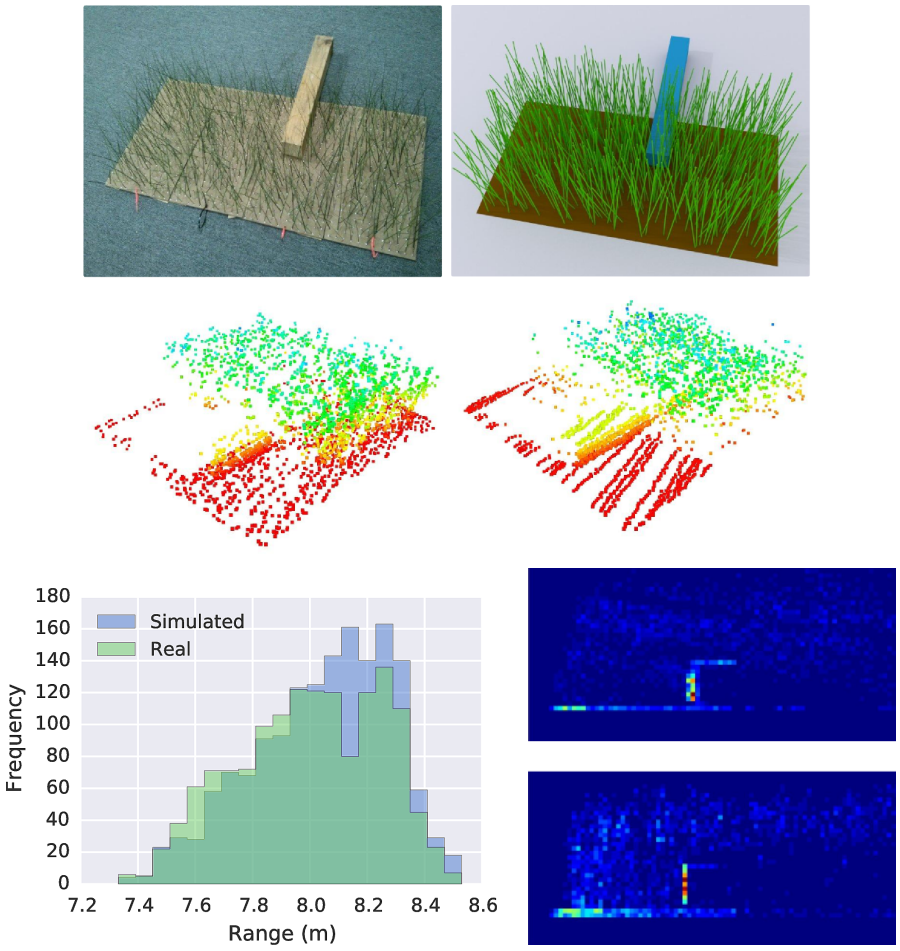


Figure 4.6: Validation of simulated laser scanner. *Top row:* the real point cloud and its simulated counterpart. *Bottom row:* the calibrated target and a render of its synthetic version.

layers followed by a fully connected network. In terms of depth, we search over architectures with one or two convolution layers (with respective pooling layers); in regards to width, we search over different number and shapes of filters for each convolutional layer, as well as the number of hidden nodes in the fully connected network. Due to time constraints, we do not explore training-related hyperparameters such as optimizer choice or weight decay. Our search space is summarized in Table 4.2.

Table 4.2: Search space for 3D CNN architecture

Hyperparameter	Values
Convolution layers (depth)	1 and 2
Filter shapes	3, 5 and 7
Number of filters	32, 64 and 128
Pooling stride	1, 2 and 4
FC hidden nodes	$2^7, \cdots, 2^{12}$

We perform a stochastic search in this space with a Tree-structured Parzen Estimator algorithm using the HyperOpt package [7]. In each case, we train a CNN with half of our dataset, and validate with the other half. For training, we optimize a negative log-likelihood loss with SGD (learning rate 0.01, momentum 0.9) for six epochs with a minibatch size of 20. For validation, we use the AUC metric.

We perform 400 hyperparameter evaluations, which took around three days when distributed across three computers. We found that many networks performed well in our task, with many obtaining virtually identical, near-perfect performance. In general, the higher-performing networks were of intermediate complexity relative to our search space. From a qualitative inspection of the lower-ranked networks, we believe that most of the smallest networks suffered from underfitting due to their low parameter count; however, for the larger networks, it is less clear, as it is possible that they were unable to fully converge with only six training epochs. Nonetheless, these first results suggested a set of networks that exhibited promising performance for our LZ task, both in terms of accuracy and computational requirements. We chose two of these architectures for further evaluation. The first, which we refer to as Voxnet-LZ1, uses one stage of convolution and pooling, with parameters $C(64, 7, 1) - P(4) - FC(512)$. The second, which we refer to as Voxnet-LZ2, uses two stages of convolution and pooling, with parameters $C(32, 7, 1) - P(2) - C(5, 64, 1) - P(2) - FC(512)$,

In terms of parameters, these networks are lightweight relative to the current state of the art networks for image data. The larger of these networks has 1.8M parameters; for comparison, AlexNet [98] has around 60M.

4.4.2 Landing Zone Detection

In this section, we further investigate how our 3D CNN architectures can be applied to the task of LZ detection. The full pipeline, for which we show qualitative results at the end of this section, is illustrated in Figure 4.7.

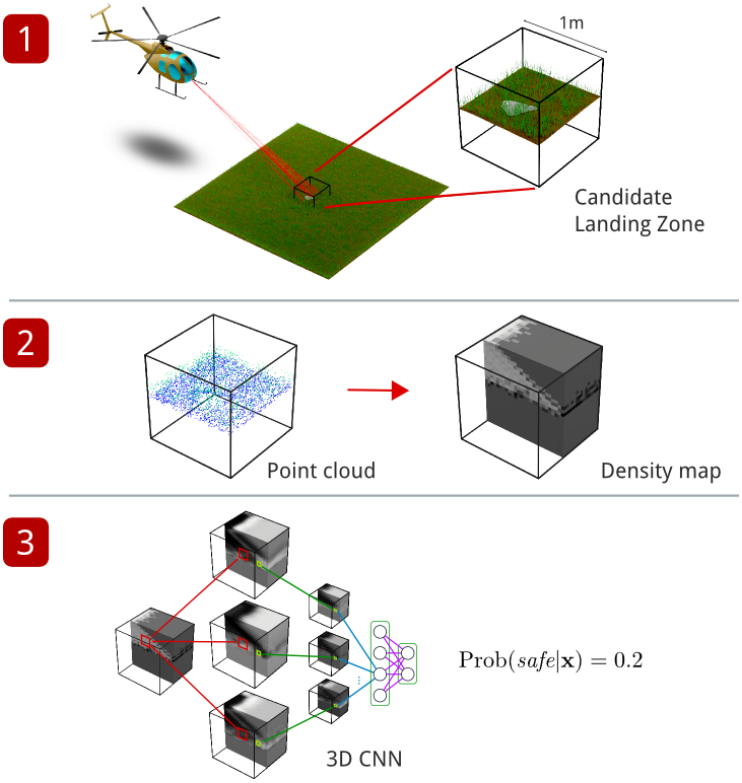
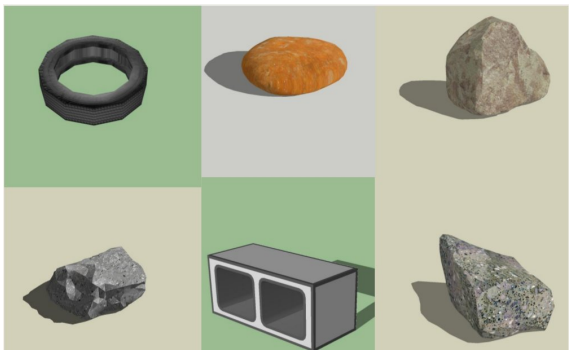


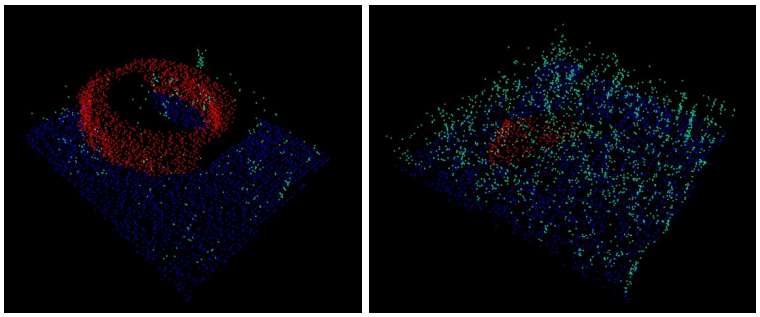
Figure 4.7: Steps in autonomous landing zone detection

Datasets

Unfortunately, acquiring and labeling real data for this scenario is difficult, as finding obstacles in cluttered point clouds is challenging even for humans. Therefore, our experiments for landing zone detection are based on two datasets; one is purely synthetic, and the other is semisynthetic. In our first set of experiments, we use a synthetic dataset created with a setup similar to that of subsection 4.4.1. The box obstacles are replaced with a selection of publicly available 3D models obtained from the Trimble 3D Warehouse[3]. In particular, we use 11 models of rocks (modelled after actual rocks, scanned by Intresto Pty Ltd[4]), a tire, and cinder blocks (Figure 4.8). The height of the objects ranges between 15 cm to 40 cm, and are all less than 50 cm across. The sensor and grass simulation parameters are kept the same. We generate 28 760 instances, with approximately half for training and the rest for validation. The training and validation sets do not have any models in common.



(a) Example CAD models



(b) Example point clouds

Figure 4.8: (a): renderings of six of the CAD models used as "obstacles". (b): Two synthetic point clouds including grass (green points), ground (blue points) and obstacles (red points).

We observed that vegetation was hard to simulate accurately, consistent with the findings of Deschaud et al. [48]. This motivated us to create semi-synthetic point clouds, consisting of real point cloud data for vegetation and ground, combined with simulated scans for solid obstacles. Since our sensor setup reports the estimated pose of the sensor at each measurement time, we

[3] https://3dwarehouse.sketchup.com
[4] https://www.intresto.com.au

can simply use our simulator with sensor rays obtained from actual point clouds, and build semi-synthetic scenes by inserting virtual obstacles in the world frame and altering rays if they intersect with the obstacle (see Figure 4.9). As before, we add noise to the simulated rays.

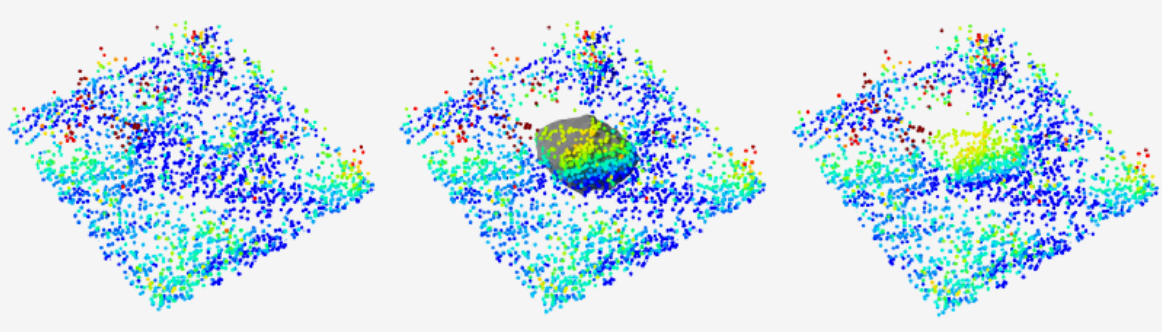


Figure 4.9: Example of semi-synthetic data generation. Left: A real point cloud of cluttered terrain. Middle: The point cloud with an inserted mesh of a rock model (modelled from a scan of a real rock). Right: The resulting semi-synthetic cloud.

To generate the semisynthetic scenes, we used lidar data from eight helicopter flights. Two of these were data collection flights from Pittsburgh, PA and six were autonomous flight missions in Quantico, VA. Figure 4.10 shows images of the vegetation in these sites. We manually selected

(a) Quantico, VA

(b) Pittsburgh, PA

Figure 4.10: Images of the vegetation present in our lidar data. Fig. (a): Vegetation in the Quantico testing site. Fig. (b): vegetation in the Pittsburgh testing site (Rock airport).

areas known to be safe for landing, and from these sampled on average 1000 patches for each flight, resulting in 21 000 patches. We then inserted random synthetic CAD obstacles with random positions and orientations in half of the patches. The CAD obstacles were the same as those used in our previous synthetic experiment.

Evaluation metrics

We use Receiver Operating Characteristic (ROC) curves to evaluate our different algorithms. For any given threshold on a detector with continuous output, we may commit two types of errors (Figure 4.11): erroneously labelling an unsafe patch as safe (a false positive), or failure to label a safe patch as safe (a false negative). A ROC curve is generated by varying this threshold and evaluating the false positive rate (FPR) and true positive rate (TPR). An ideal algorithm would always have TPR equal to 1, and random chance would have TPR=FPR. Note that we are implicitly

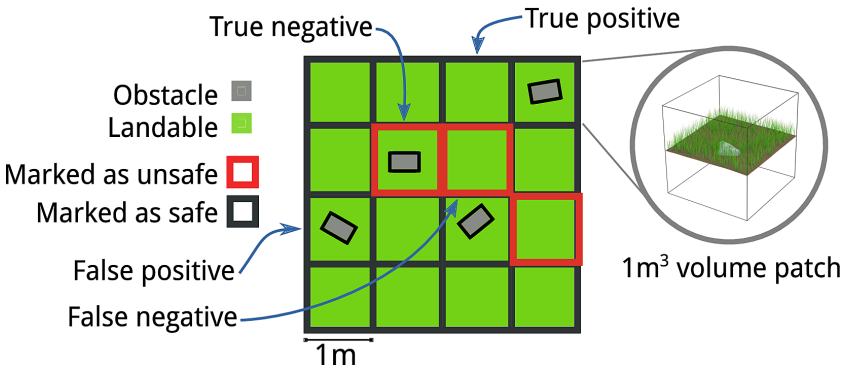


Figure 4.11: Possible outcomes for landing zone safety prediction. The gray rectangles represent obstacles.

assuming each volume has a 0.5 prior probability of being safe; this can be easily changed to incorporate prior knowledge by scaling the output appropriately [11].

Baselines

Our first baseline is based on the residuals of a robust plane fit, as proposed in Scherer [152] and similar geometric methods. The sum of the residuals (clipped to a maximum of $0.1\,\mathrm{m}$ for robustness) was used as the continuous output in the ROC curve.

The second baseline is a Random Forest classifier [14], using the implementation of `scikit-learn` [138]. It is trained with the same raw volumetric data as our networks. We used 20 trees with no maximum depth. Random forests are known to be robust and usually effective classifiers that work well with high-dimensional data. This baseline has no built-in invariance to spatial shifts.

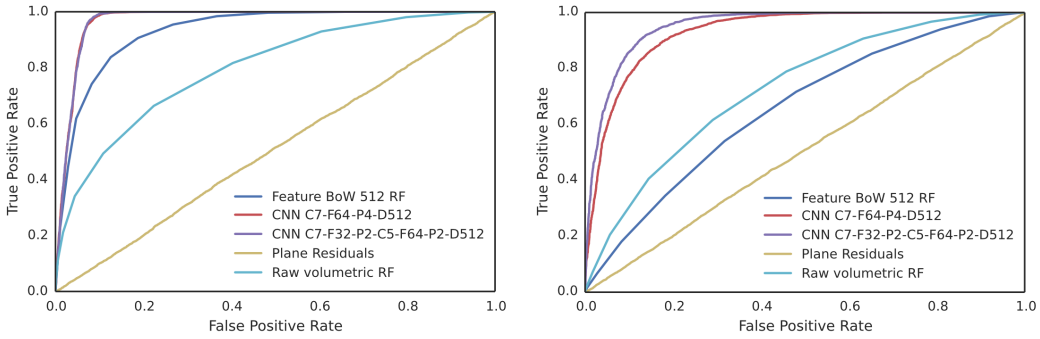
The third baseline is a system built with various well-known features from the literature on point cloud classification, similar to those of chapter 3. These include three spectral shape features from Lalonde et al. [103], three directional features and three shape features from Shapovalov et al. [158], and range variance features from Macedo, Manduchi, and Matthies [116]. We calculate each feature on a per-voxel basis, with $3 \times 3 \times 3$ voxel spatial smoothing for the spectral features. We carefully tuned these features to work well in this data, as they were part of our first approach to solve this problem (see chapter 3).

We construct a K-means codebook with 512 words from $1/4$ of the training data and represent each volume with a softly-quantized Bag of Words (BoW) [37]. Finally, we classify the BoW with a random forest classifier trained using the same parameters as before. This approach is similar to approaches that until recently were considered state-of-the-art for various tasks in computer

vision. While the BoW ignores any spatial structure in the data, the features encode some local spatial context (*e.g.*, height relative to the ground).

Results

We show the results of the baselines and our VoxNet models on our synthetic grass and obstacle dataset in Figure 4.12a and Table 4.3.



(a) Synthetic obstacles ROC

(b) Semisynthetic obstacles ROC

Table 4.3: Area under curve (AUC) of evaluated methods

Method	AUC Synth	AUC Semisynth
Feature BoW 512 RF	0.931	0.66
VoxNet-LZ1	0.97	0.93
VoxNet-LZ2	0.97	0.95
Plane Residuals	0.51	0.50
Raw volumetric RF	0.80	0.73

In the synthetic obstacles dataset, the two CNN approaches take the lead and perform almost indistinguishably. This suggests they are learning similar hypotheses despite their different architecture, or that they are reaching some limit related to the dataset. The plane residuals perform barely above chance. This is due to the fact that by construction, our clouds are relatively dense and always have at least some clutter.

Results in the semisynthetic dataset are shown in Figure 4.12b and Table 4.3. As before, the plane residual method is no better than chance. The two random forest methods are better, but are clearly outperformed by the CNN methods. Out of these, the deeper architecture, VoxNet-LZ2, obtained slightly better performance, suggesting that the additional depth is beneficial in this more challenging scenario.

Qualitative results Some representative successes and failures from the semisynthetic dataset are shown in Figure 4.13. Our method sometimes results in false negatives when the vegetation is dense enough to resemble rocks, or results in false positives when the obstacle is very small or mostly occluded.

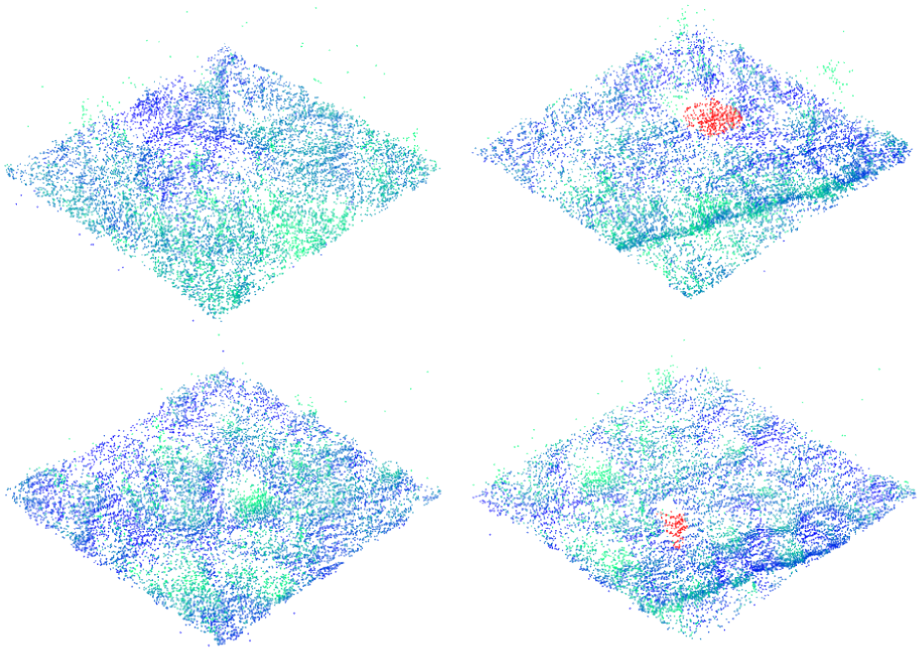
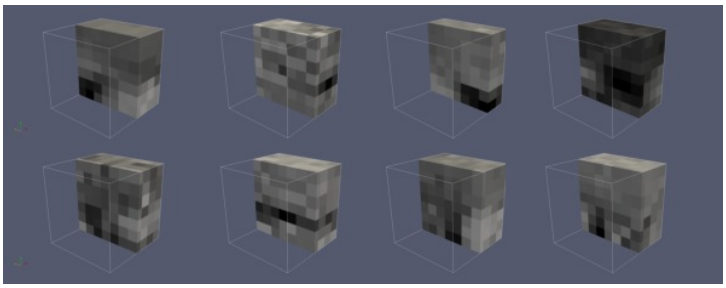
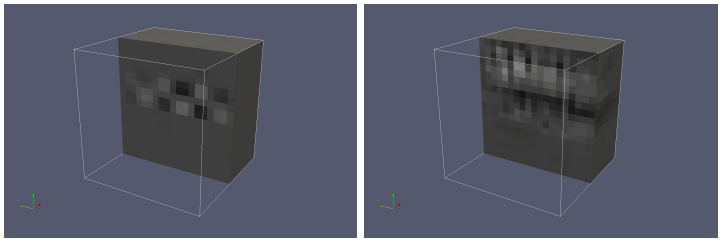


Figure 4.13: Example outputs for landing zone safety prediction. Top: two correct landing zone safety assessments (true positive and true negative). Bottom: Two failures (false negative and false positive). Obstacles are shown in red. In the first failure case, there are several dense bushes which are similar to rocks. In the second, only a very small portion of the obstacle is visible.

We also apply two visualizations commonly applied to CNNs for image data. In the first, we examine the weights of the first convolutional layer of VoxNet-LZ1 (Figure 4.14a). We visualize a slice of selected filters, visualized as a 3D grid, for the network trained on the synthetic and semisynthetic datasets. They exhibit similar spatial structure for both datasets, and appear to be specialized to detect corner, blob and plane-like structures.



(a) Selected filters from Voxnet-LZ1 on the synthetic (top row) and the semi-synthetic (bottom row) datasets, where darker means a higher value.



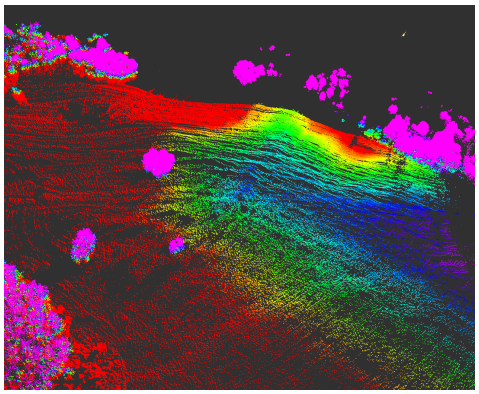
(b) Cross-sections of hallucinated unsafe and safe volumes from VoxNet-LZ1.

Visualizing the higher levels of the network is less straightforward. Our second visualization uses the technique from [160] to hallucinate an "ideal" input for each category predicted by the network by backpropagation on the input. We show hallucinated volumes from VoxNet-LZ1, trained on the synthetic dataset, in Figure 4.14b. We observe the ideal unsafe volume has multiple box-like sets of planes, whereas the ideal safe volume has visible ground and free space.

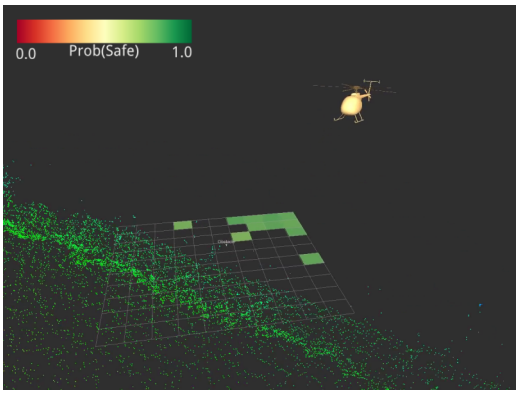
Real-time LZ detection pipeline

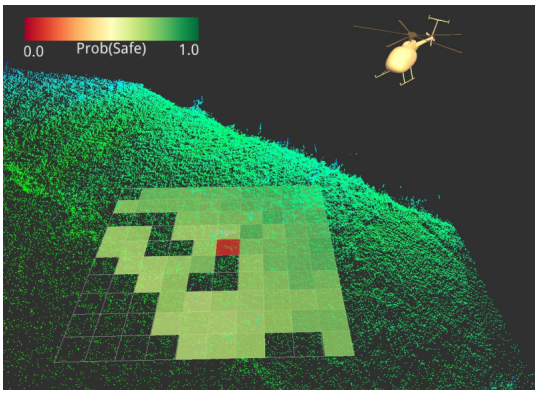
We built a pipeline based on VoxNet-LZ2 capable of performing real-time LZ detection over an area of interest (we use an area of $10 \times 10\,\mathrm{m}^2$) by partitioning it into smaller patches of $1 \times 1\,\mathrm{m}^2$ and processing each individually. Our pipeline incrementally updates the occupancy grids from the lidar point clouds and performs inference after each update, so the vehicle can improve its assessment of LZs by accumulating a higher-density coverage of the ROI. Figure 4.15 shows a screen capture of this pipeline operating in a semi-synthetic scenario, where we insert a synthetic obstacle in flight data captured from a real mission.

(a) Satellite view of LZ region with overlaid trajectory

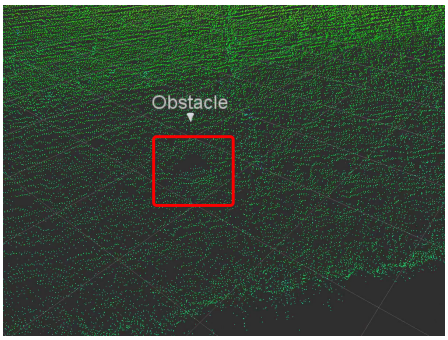


(b) Wide area view of point cloud as the helicopter begins landing approach





(c) Sequential screen captures of on-line LZ safety evaluation



(d) Close-up of inserted rock obstacle detected by the system

Figure 4.15: Screen capture of integrated system operating in real time. The system evaluates the safety of a $10 \times 10\,\mathrm{m}^2$ area with a resolution of $1 \times 1\,\mathrm{m}^2$. The video is available at https://youtube.com/playlist?list=PLeg9sULe3rSlz3xbjx3WrNk5FS2J-DCWw.

Timing

Depending on the parameters of the network, training for six epochs takes between two to six hours on our Core 2 DUO equipped with a 3GB GTX580 GPU. On the other hand, labeling a $1\,\mathrm{m}^3$ patch takes less than $5\,\mathrm{ms}$, and ray tracing (which is done on the CPU) to compute hit/passes and density on 3000 points takes less than $1\,\mathrm{ms}$ per 1000 points for a $400 \times 400 \times 40$ voxel grid. While not a fair comparison, as it runs on the CPU, the BoW algorithm by itself takes around $200\,\mathrm{ms}$ per volume. Around half of the time is spent performing feature extraction, and the other half is spent performing quantization.

4.4.3 Object Recognition

In this section, we investigate the application of our 3D CNN architecture to more general object recognition tasks with data from three different sources (lidar, RGBD and CAD data). We also show qualitative results in a live demonstration with data from yet another source (time-of-flight sensor).

In addition, we introduce a modified, more lightweight, version of the VoxNet-LZ2 architecture, which we simply refer to as VoxNet.

Datasets

Our datasets use 3D data from three different sources: lidar point clouds, RGBD point clouds, and CAD models. Figure 4.16 shows example instances from each source.

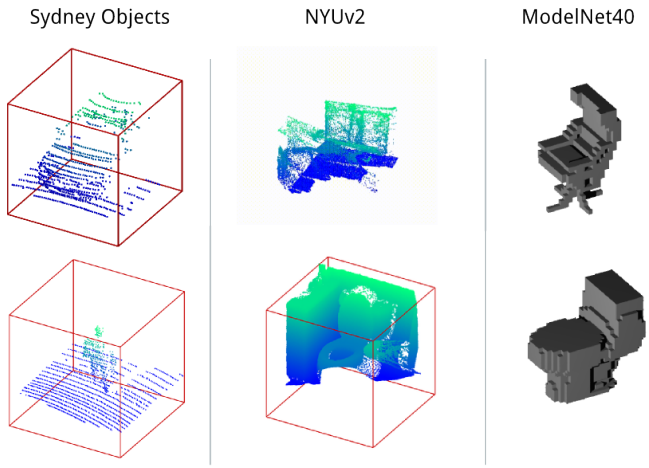


Figure 4.16: Object instances from the Sydney Objects, NYUv2 and ModelNet40 datasets. *Left*: The Sydney Objects dataset consists of lidar scans captured in urban scenes. *Middle*: The NYUv2 dataset contains RGBD point clouds of various indoor locations, captured with a Kinect sensor. *Right*: ModelNet40 is a dataset of CAD models of various common objects, mostly furniture. We use voxelized versions of these models.

Lidar data — Sydney Urban Objects: The Sydney Urban Objects Dataset[5], used in Deuge et al. [49], contains labeled Velodyne lidar scans of 631 urban objects in 26 categories. We chose this dataset for evaluation as it provides labeled object instances and the lidar viewpoint, which is used to compute occupancy. When voxelizing the point cloud, we use all points in a bounding box around the object, including background clutter. This makes the task more challenging, but also more realistic relative to real world scenarios. We follow the protocol employed by the dataset authors, who report the average class-weighted F1 score over four training/testing splits. For this dataset, we perform augmentation at training and test time with 18 rotations per instance. We report the average F1 score, weighted by class support, for a subset of 14 classes over four standard training/testing splits.

CAD data — ModelNet: The ModelNet datasets were introduced by Wu et al. [184] to evaluate 3D shape classifiers. ModelNet40 has 151,128 3D models classified into 40 object categories, and ModelNet10 is a subset based on classes that are found frequently in the NYUv2 dataset [129]. The authors provide the 3D models as well as voxelized versions, which have been augmented by 12 rotations. We use the provided voxelizations and train/test splits for evaluation. In these voxelizations the objects have been scaled to fit a $24 \times 24 \times 24$ grid. We report the accuracy averaged per class.

RGBD data — NYUv2: Wu et al. also evaluate their approach on RGBD point clouds obtained from the NYUv2 dataset [129]. We use the train/test split provided by the authors, which uses 538 images from the RMRC challenge[6] for training, and the rest for testing. After selecting the boxes sharing a label with ModelNet10, we obtain 1386 testing boxes and 1422 training ground truth boxes. Wu et al. report results on a subset of these boxes with high depth quality, whereas we report results using all boxes. For this dataset, we compute our own occupancy grids. However, to make results comparable to Wu et al. we do not use a fixed voxel size; instead, we crop and scale the object bounding boxes to $24 \times 24 \times 24$, likewise, we use 12 rotations instead of 18. Unlike Wu et al., we do not use a per-pixel object mask to remove outlying depth measurements from the voxelization. As in the Sydney Objects dataset, we keep all points in a bounding box around the object; from the voxelization, possibly making the task more difficult due to presence of clutter in the voxelized data.

Architecture

In these experiments, we use a modified version of the VoxNet-LZ2 architecture from subsection 4.4.2, which we simply refer to as VoxNet. Following the recent trend towards smaller convolutional filters [161], this version reduces the size of the layers in the first and second convolutional filters, as well as the number of hidden neurons in the fully connected layer. To reduce the computational requirements, we add striding with factor 2 to the first convolutional layer and remove the first pooling layer. We found this network, which was also among the top-performing networks in subsection 4.4.1, to be slightly faster and easier to learn than VoxNet-LZ2. In the convention of subsection 4.4.1, the architecture of VoxNet is $C(32, 5, 2) - C(32, 3, 1) - P(2) - FC(128)$ (Figure 4.3).

[5] http://www.acfr.usyd.edu.au/papers/SydneyUrbanObjectsDataset.shtml
[6] http://ttic.uchicago.edu/~rurtasun/rmrc/

Rotation augmentation We study four different cases for rotation augmentation, depending on whether it is applied or not at train time (as augmentation) and test time (as voting) for the Sydney Objects and ModelNet40 datasets. For the cases in which no voting is performed at test time, a random orientation is applied on the test instances, and the average over four runs is reported. For the cases in which no training time augmentation is performed, there are two cases. In ModelNet40, we select the object in a canonical pose as the training instance. For Sydney Objects, this information is not available, and we use the unmodified orientation from the data. Table 4.4a shows the results. They indicate that training time augmentation is more important. As suggested by the qualitative example above, the network learns some degree of rotational invariance, even if not explicitly enforced. However, voting at training time still gives a small boost. For ModelNet40, we see a large degradation of performance when we train on canonical poses but test on arbitrary poses, as expected. For Sydney Objects there is no such mismatch, and there is no significant degradation in this case. Since rotation augmentation seems consistently beneficial, in the rest of the results we use VoxNet with rotation augmentation at both training time and test time.

Table 4.4: Effects of rotation augmentation and occupancy models on the Sydney and ModelNet benchmarks

(a) Effects of rotation augmentation at training and test time

Training Augm.	Test Augm.	Sydney F1	ModelNet40 Acc.
Yes	Yes	**0.72**	**0.83**
Yes	No	0.71	0.82
No	Yes	0.69	0.69
No	No	0.69	0.61

(b) Effect of occupancy model

Occupancy	Sydney F1	NYUv2 Acc.
Density grid	**0.72**	**0.71**
Binary grid	0.71	0.69
Hit grid	0.70	0.70

Occupancy model We also study the effect of the Occupancy Grid representation in Table 4.4b. We found VoxNet to be quite robust to the different representations. Against expectations, we found the Hit grid to perform comparably or better than the other approaches, though the differences are small. This is possibly because any advantage provided by differentiating between free space and unknown space is negated by the extra viewpoint-induced variability of Density and Binary grids relative to Hit grids. By default, we will use Density grids in the experiments below.

Resolution For the Sydney Object dataset we evaluated VoxNet with voxels of size $0.1\,\mathrm{m}$ and $0.2\,\mathrm{m}$. We found them to perform almost indistinguishably, with an F1 score of 0.72. On the other

Table 4.5: Comparison with baselines in the Sydney Object, ModelNet and NYUv2 benchmarks

(a) Sydney Object Dataset (Average F1)

Method	Avg. F1
UFL+SVM[49]	0.67
GFH+SVM[31]	0.71
VoxNet	**0.72**

(b) ModelNet (Average accuracy)

Method	ModelNet10 Acc	ModelNet40 Acc
ShapeNet[184]	0.84	0.77
VoxNet	**0.92**	**0.83**

(c) NYUv2 (Average accuracy)

Method	NYUv2 Acc	ModelNet10→NYUv2 Acc
ShapeNet[184]	0.58	**0.44**
VoxNet	**0.71**	0.34

hand, fusing both with the multiresolution approach slightly outperformed both with a score of 0.73. However, as this approach is relatively slow, we will use $0.1\,\mathrm{m}$ by default.

Results

We compare VoxNet against publicly available results in the literature for each dataset.

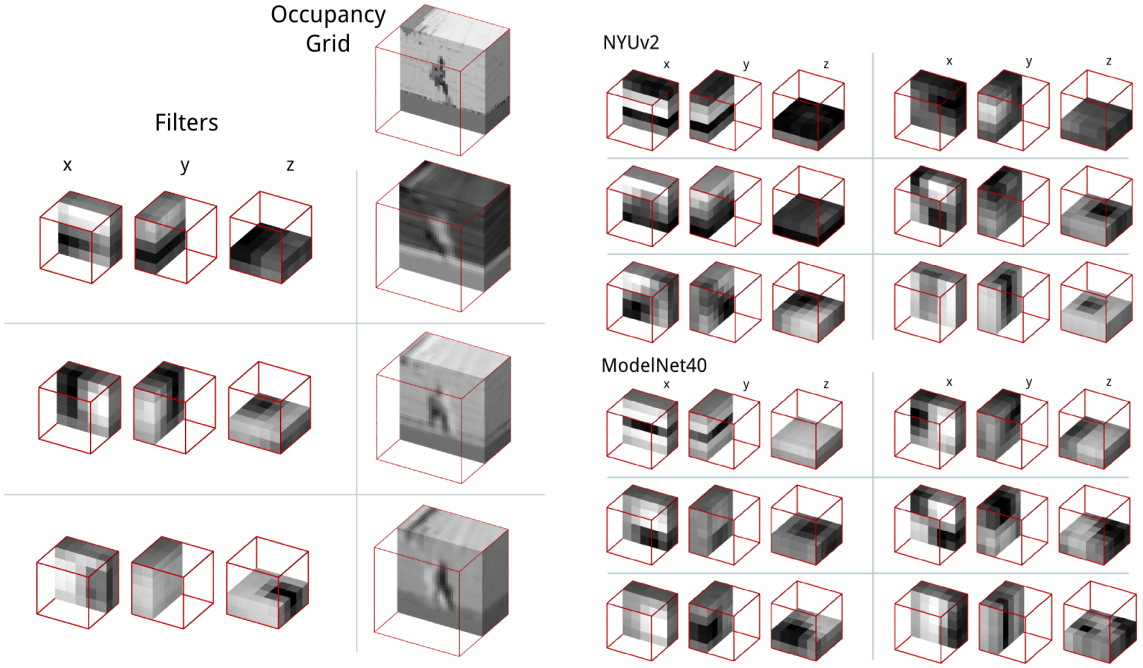
Table 4.5a shows VoxNet with the best approach from Deuge et al. [49], which combines an unsupervised form of Deep Learning with SVM classifiers, and Chen et al. [31], which designs a rotationally invariant descriptor and classifies it with a nonlinear SVM. We show a small increase in accuracy relative to these approaches. Moreover, we expect our approach to be much faster than approaches based on nonlinear SVMs, as these do not scale well to large datasets.

Table 4.5b compares VoxNet with the ShapeNet architecture proposed by Wu et al. [184] in the task of classification for ModelNet10 and ModelNet40. Shapenet is also a volumetric convolutional architecture; it is trained generatively with discriminative finetuning, and also employs rotation augmentation for training. It has over 12.4 million parameters, while VoxNet has less than 1 million. In these datasets VoxNet outperforms ShapeNet by a fairly large margin.

Table 4.5c compares VoxNet with ShapeNet in the NYUv2 dataset and in the task of classifying the NYUv2 dataset with model trained on ModelNet10. While VoxNet significantly outperforms ShapeNet in NYUv2 task, it performs significantly worse in the cross-domain classification task. We speculate that the simpler architecture of VoxNet may result in better generalization when using purely discriminative training, but may be less capable of dealing with domain shift.

Qualitative results To get some insight into what the network is learning, we visualize some filters from the first convolutional layer and the features they extract in Figure 4.17. The filters in this layer seem to encode primitives such as edges, corners, and "blobs".

Figure 4.17a depicts cross sections of some learned filters from the input layer and the corresponding feature maps learned from the input in the Sydney Objects dataset. The filters in this layer are similar across datasets and seem to encode primitives such as edges, corners, and "blobs". Qualitatively, the filters from VoxNet appear to be smoother than those of the VoxNet-LZ1 and VoxNet-LZ2 variants from subsection 4.4.2. One reason for this may be their smaller size ($5 \times 5 \times 5$ versus $7 \times 7 \times 7$) acts as a regularizer, forcing the filters to learn more generic features.



(a) (b)

Figure 4.17: Visualization of first layer filters and feature maps in object detection datasets. Fig. (a) shows three first-layer filters (one per row, with three cross-sections each) learned from the Sydney Objects database, along with a cross-section of the corresponding feature map on the right. Each filter extracts different spatial structure from the occupancy grid. In Fig. (a), we show six first-layer filters learned from NYUv2 and ModelNet40, with three cross-sections each. Qualitatively, the filters extracted across different datasets appear to capture similar spatial structures.

As we discuss in section 4.3, yaw rotations may cause large shifts in voxelized appearance, an issue we attempt to counteract with data augmentation. A natural question is whether the network trained thus learns some degree of rotational invariance. Figure 4.18 shows an example supporting this hypothesis, where the two fully connected layers show a highly (but not completely) invariant response across 12 rotations of the input.

Interactive demonstration with a portable ToF sensor A live, interactive demonstration of VoxNet was demonstrated at the 2017 Robot Week at Carnegie Mellon University Figure 4.19a as

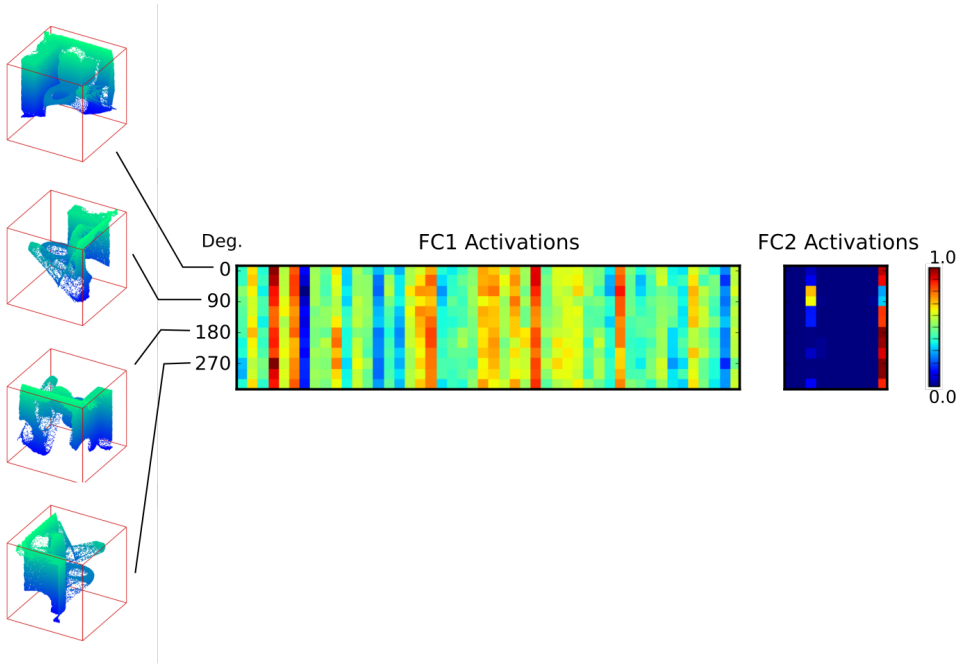
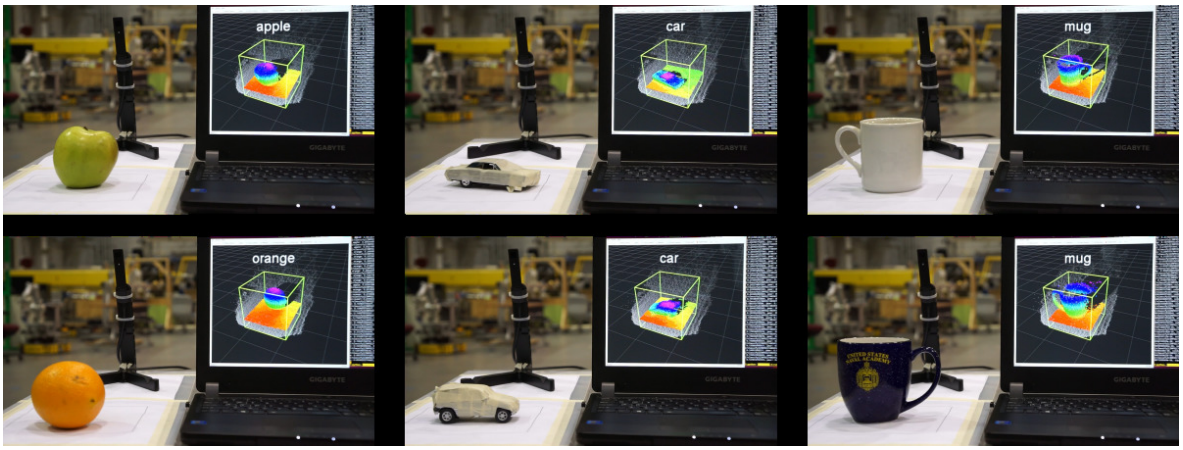


Figure 4.18: Neuron activations for the two fully connected layers of VoxNet with an RGBD point cloud of class *toilet* from NYUv2, across 12 different orientations. For the first fully connected layer, only the first 48 features are shown. Each row corresponds to a rotation around z, and each column corresponds to a neuron. The similar activation magnitude along each column shows approximate rotational invariance. The neurons in the right correspond to output classes, with the last column corresponding to the *toilet* class. Near $90°$, the object becomes confused with a *chair* (third column); however, by voting across all orientations we obtain the correct answer.

well as the 2018 Robotics: Science and Systems conference in Pittsburgh. The demonstration used a PMD PicoFlexx, a portable time-of-flight sensor that operates with a similar principle to lidar, and a laptop equipped with a GT980M GPU. With this setup, VoxNet was able to classify and display predictions for a variety of object categories in real time (20 FPS). The object categories included a variety of objects including fruits, different types of toys and other objects found in an office environment, such as mugs and markers (Figure 4.19b). A video of this demonstration can be seen at https://youtu.be/KAB11FrQz_Q.

(a) Public interactive demonstration during National Robotics Week 2017, Carnegie Mellon University



(b) Screen captures of demonstration. Video can be found at https://youtu.be/KAB11FrQz_Q.

Figure 4.19: Interactive VoxNet demonstration

4.5 Summary

In this chapter, we describe a novel architecture for classification of 3D point clouds using occupancy grids with a 3D CNN. The system obtains state-of-the-art accuracy in various benchmarks

while taking only a few ms per volume. Qualitative results suggest that the 3D CNNs learn to extract useful features and invariant representations for their task, similarly to their 2D counterparts. We expect that this architecture can be useful in many different tasks involving 3D point clouds and extended in multiple ways.

However, this chapter is only a first step showing the feasibility and usefulness of learning deep representations of spatial information with 3D CNNs. There are several issues and questions that this work does not address. Many of these issues have been researched in the years since the original publications documenting this work.

For example, our volumetric representations are relatively coarse and low-resolution due to their relatively high memory requirements when implemented naively. This is limiting in several ways. For example, for any given fixed volume there is a trade-off in resolution versus scale invariance when voxelizing a point cloud; if the point cloud is not scaled, then the voxelized point cloud may be too low resolution, or the ROI may not fit in the volume (Figure 4.20). If the point cloud is scaled, then object categories that are similar in shape but typically different in size may be confused. Since our initial publications, several options have been proposed to address this issue in a volumetric framework, such as sparse 3D CNNs [71, 36] and hierarchical versions of 3D CNNs [149].

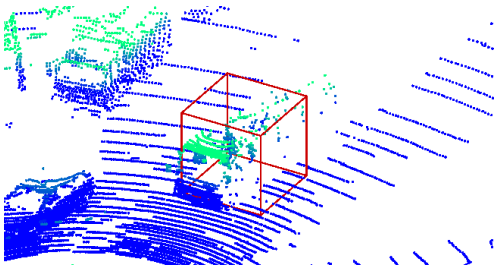


Figure 4.20: When using a fixed-size bounding box to crop point clouds, large objects such as the bus may not fit inside it, making classification difficult. On the other hand, if the point cloud is scaled to fit inside the bounding box, the bus may become difficult to distinguish from smaller but similarly-shaped vehicles.

Another question is what is the most effective way of encoding spatial information in 3D volumes. We have favored occupancy representations under the hypothesis that knowledge about free space and occupied space is useful for classification; however, we have seen that merely encoding occupied space (as "hit grids") can be surprisingly effective, possibly due to being less affected by capture viewpoints. We have also not considered other common options, such as signed distance functions, that may be effective in encoding more spatial information than hit grids while being less affected by viewpoint.

It is not a given that volumetric representations are the best option for 3D semantic perception tasks. As we have mentioned, they have some drawbacks, and other options (with different tradeoffs) are possible; for example, shortly after this work, Su et al. [165] proposed using 2D CNNs applied to multiple 2D projections of 3D shapes as an alternative. Another popular line of work has proposed using 3D point clouds directly using recurrent neural networks [139] and graph neural networks [180]. These and many other works have steadily improved on the state of

the art for the benchmarks we use in our experimental evaluation[7].

Despite the limitations of the original formulation of VoxNet, it has seen a variety of interesting applications and extensions, including generative shape modeling [15], 3D semantic segmentation [84], robotic grasp prediction [33, 134], hand pose estimation [66], 3D object detection [54], and many others.

While this chapter has shown that 3D point clouds — whether they are analyzed with VoxNet or more recent approaches — are a useful data source for many semantic perception tasks, their sparsity and lack of color information makes them unsuitable for many applications. This makes images — which lack direct 3D information, but usually encode color and are comparatively dense — an interesting source of complementary information for semantic inference. In the next chapter, we investigate how to use both of these modalities for semantic mapping.

[7] A compilation of results for ModelNet40 can be found in https://paperswithcode.com/sota/3d-point-cloud-classification-on-modelnet40.

Chapter 5

Multimodal semantic mapping with image and point cloud data

> PICTURE, n. A representation in two dimensions of something wearisome in three.
>
> Ambrose Bierce
> *The Devil's Dictionary*

In the past two chapters, we explored methods that showed that point cloud data can be used effectively for semantic as well as spatial inference. However, in many scenarios, it may be difficult to acquire point clouds with the density and accuracy necessary to discriminate between classes with similar geometry. Moreover, many types of surfaces and objects are easier to discriminate by properties other than their shape, such as color or texture.

Meanwhile, it is common for vehicles equipped with high-quality range sensors to also have one or more cameras at their disposal, given their comparatively low cost and footprint. Camera imagery, while lacking in direct 3D information, provides high-resolution texture and color cues that are usually not captured by range sensors. As shown by the literature in computer vision, this information is extremely valuable for semantic inference.

This motivates us to build semantic mapping systems that use both range sensors and cameras, in order to benefit from the complementary information provided by each. In this chapter, we investigate the joint use of these sensing modalities in two contributions:

- We investigate how to use point cloud and image data jointly for semantic inference with a novel approach that learns how to fuse these two modalities with a hybrid 2D-3D CNN architecture. While this approach shows promising results in offline semantic segmentation benchmarks, our implementation is too computationally intensive for use in real-time semantic mapping.
- We propose a decoupled approach for multimodal semantic mapping based on the premise that we can achieve real-time operation with acceptable accuracy by using each modality independently in a way that takes advantage of their relative strengths. In this approach, we use image data for semantic inference and point cloud data for spatial mapping, and fuse the respective outputs of these modules with a simple projection-based method.

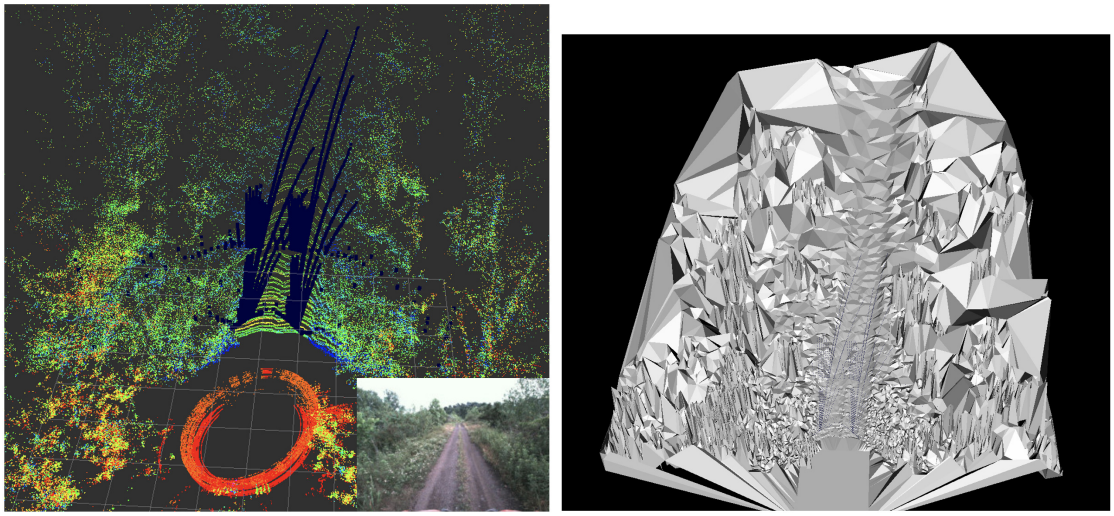
Multimodal semantic mapping with point cloud and image data has a wide range of applications, as sensor setups that provide both point cloud and image data have become increasingly popular in robotics and other applications. The work in this chapter uses data provided by a lidar and a camera sensor; however, the ideas are also applicable to multimodal data originating from structured light (RGBD) sensors, as well as stereo sensors.

However, the work in this thesis was originally motivated by a specific application: autonomous off-road navigation for an All-Terrain Vehicle (ATV). As we evaluate both of our contributions in the context of this application, we will provide additional background to this task in the next section.

5.1 Helping an autonomous All-Terrain Vehicle find its way

The past few years have seen a surge of activity in self-driving research. The bulk of recent research has focused on urban driving scenarios, despite the fact that many influential systems in the area were developed for off-road vehicles [63, 171, 175].

In this chapter, we revisit the off-road driving scenario, and in particular, one of its most basic problems: finding traversable paths. Our platform, Erik (Figure 5.2) is a modified All-Terrain Vehicle (ATV) equipped with high-quality INS, lidar and camera systems that allow a state-of-the-art mapping and planning framework [182] to build accurate mesh-based maps in real time; Figure 5.1 shows an example of a point cloud and its corresponding mesh.



(a) (b)

Figure 5.1: Example point cloud (a) and its corresponding mesh representation (b) used for spatial mapping in the ATV. Potential trajectories considered traversable according to geometric criteria are visualized in blue. Because of the vegetation in the middle and edges of the trail, the planner considers considers most of the trail untraversable. Figure courtesy of Mesh Robotics, LLC.

However, as we have observed in prior chapters, relying solely on geometric information leads to disappointing results for autonomous navigation in off-road environments. For example, in

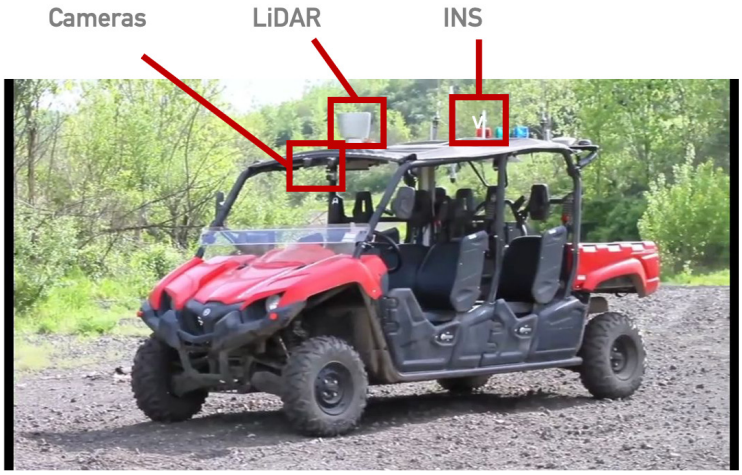


Figure 5.2: The ATV platform and its sensors

Figure 5.1 the vegetation in the middle and edges of the trail causes the mesh representation of the trail to appear too rough to traverse at high speeds, even if this is low, sparse vegetation that does not pose a risk for the ATV. Figure 5.3 shows more examples of challenging scenarios for a purely geometric representation. All of these may lead to suboptimal, even dangerous, decisions in path planning. Similar observations have been made many times before in the context of off-road robotics, *e.g.*, [90, 118, 171, 85].

(a) Grass-covered trail (b) Narrow trails surrounded by tall vegetation (c) Muddy trails

Figure 5.3: Challenging scenarios for path following

As in our earlier chapters, we investigate semantic mapping to counter this problem. Our goal is to create semantic maps that encode relevant geometric (*e.g.*height, roughness) and semantic (classes such as trail, grass, obstacle, etc.) aspects of the vehicle's surroundings in real time, and which can be used by a planning system to generate sound paths through terrain that is deemed untraversable based solely on its geometry. Unlike our earlier chapters, in this chapter we leverage the availability of image data in addition to lidar data for the construction of our maps.

Like before, we do not address the 3D localization and mapping problem, *i.e.*, we assume that we have access 3D point clouds and camera poses that are registered in a common frame. However, there are still various challenges we must solve to build an effective semantic mapping system. Our system must be able to accurately discriminate among classes that exhibit high variation in appearance according to various factors; Figure 5.4 shows that even within a single day and

location, terrain appearance may vary widely. Moreover, the system must operate at high rates to guide high-speed navigation.

Figure 5.4: Trail images captured within a single day at the Gascola site near Pittsburgh, PA. Despite being captured within a time span of a few hours and a distance of less than a square mile, the different categories of interest show a a high diversity in appearance due to intrinsic factors (*e.g.*, different types of soil) or extrinsic (variation in lighting and imaging parameters).

In the next two sections, we will describe the two contributions of this chapter for semantic mapping with point cloud and image data, and evaluate both in the context of autonomous off-road navigation.

5.2 Joint 2D-3D CNN for multimodal semantic segmentation

As illustrated by many success stories in computer vision, image data is a valuable cue for semantic inference. Meanwhile, as we have shown in the last two chapters, point cloud data can also be used to good effect for semantic classification and segmentation. In both cases, deep learning, and specifically, CNNs, have been successfully used to learn discriminative feature representations.

Given the different type of information encoded in images and point clouds, this naturally leads to the question on whether combining both modalities can be more useful for semantic inference over either alone, and if so, how can this be achieved.

There is reason to believe that the combination of image and point cloud data can outperform either alone. Each of these modalities capture different aspects of the environment that are likely valuable for different semantic inference tasks. For example, the high-resolution color and texture data provided by image data is useful to discriminate between rough terrain and vegetation, whereas both may appear similar in sparse point clouds. On the other hand, the geometric information captured in point clouds is inherently more invariant to lighting and color variations, which can be useful to provide robust inference across different seasons and even times of the day. In addition to the value of the cues each of these modalities provide when considered independently, joint analysis of image and point cloud data may reveal patterns that are not evident in either modality

by itself. For example, depth cues from point cloud data may help disambiguate changes in image appearance caused by varying distance to the camera.

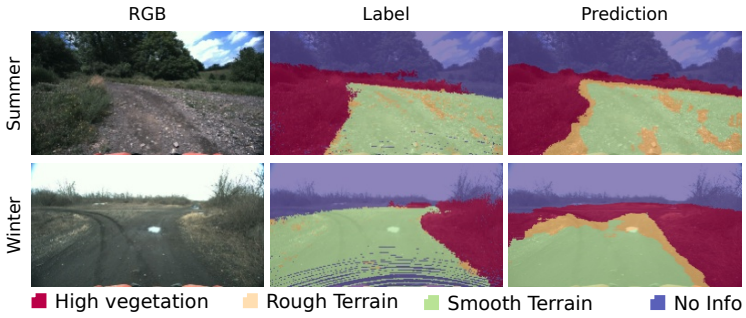


Figure 5.5: An image-based CNN [5] trained on a sunny summer dataset (top row) cannot predict robustly when a test dataset has severe appearance variations, such as on a cloudy winter dataset (bottom row).

Figure 5.5 shows a motivating example from our target application, autonomous off-road driving. In this example, an image-only approach to semantic segmentation is adversely affected by appearance variations induced by seasonality and lighting.

A multimodal approach using image and point cloud data creates opportunities for learning-based approaches such as CNNs to take advantage of their complementary characteristics. However, it is not obvious how to build an architecture that is capable of learning how to fuse the information from each modality.

In this contribution, we propose a solution in the form of a novel hybrid 2D-3D CNN architecture. We discuss this architecture in subsection 5.2.1. We then evaluate this architecture for semantic segmentation in the context of autononous off-road driving in section 5.2.6.

5.2.1 Architecture overview

Our proposed network comprises a 3D CNN for point cloud data, based on the work from chapter 4, a 2D CNN for image data, based on recent work in fully convolutional networks for image segmentation [113, 137], and projection modules that enable the propagation of information from the 3D CNN to the 2D CNN.

The inputs to the network consist of a RGB image and a point cloud encoded as a 3D volumetric grid. We assume that we have the necessary information to project any 3D point in the frame of the point cloud data to a 2D position in the image frame, *i.e.*, that we have calibrated the intrinsic and extrinsic parameters of the sensors.

The output of our network is a 2D feature map, which can be adapted for different tasks. Here, we use this map for 2D semantic segmentation. To this end, we adopt the method of Long, Shelhamer, and Darrell [113], where the output is a feature map consisting of K channels, each corresponding to one of the classes to be segmented. We predict four semantic classes: (“High vegetation”, “Rough terrain”, “Smooth terrain”, and “No Info”).

Figure 5.6 shows a high-level view of this architecture. The next sections describe each component in this network.

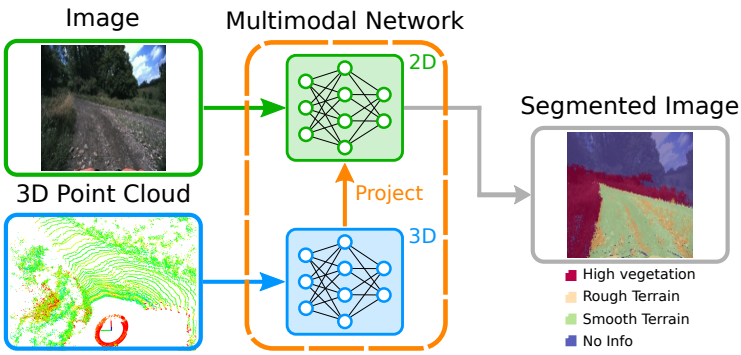


Figure 5.6: Our multimodal network takes inputs of an image and a 3D point cloud. Our network learns and combines 2D-3D features; and outputs a segmented image. Point clouds are false colored by intensity.

5.2.2 2D image network

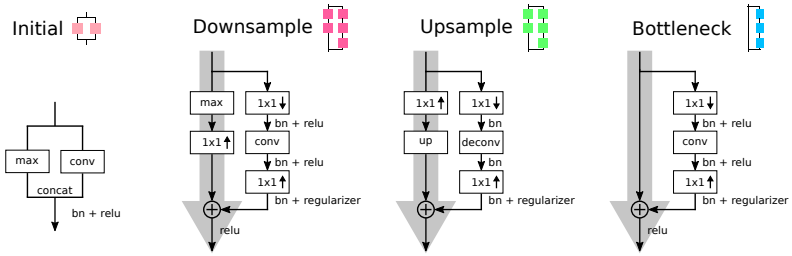


Figure 5.7: Modules used in our network, based on ENet [137]. *max*: max-pooling layer with non-overlapping 2×2 windows. *up*: upsampling layer with a factor of 2. *conv*: A regular, dilated, or asymmetric convolution layer. *bn*: batch normalization. *regularizer*: spatial dropout. 1×1 *with down or up arrow*: A 1×1 convolution to reduce or expand channels.

The goal of the image network is to learn 2D feature representations θ^{2D} from images that can help the overall architecture make robust predictions. The network should have a good segmentation performance, but also have a fast prediction time and a small footprint to allow its use in a real-time autonomous system. In this work, we design the network based on ENet [137], which has demonstrated similar performance to existing recent models (*e.g.*, SegNet [5]) but with faster inference and a lower parameter count. ENet follows an encoder-decoder structure and is built with a set of blocks shown in Figure 5.7. In the encoder, feature maps are progressively downsampled to allow the convolutional filters in each stage to capture a wider spatial context; in the decoder, the feature maps are progressively upsampled until they reach the original resolution. Throughout, residual connections [80] are added to facilitate the propagation of gradients during training. The network, as used in this work, is depicted in the top half of Figure 5.9.

5.2.3 3D point cloud network

Similarly to the image network, the point cloud network extracts multiple layers of 3D features θ^{3D} for our segmentation task. For our experiments, we use a 3D version of the image network

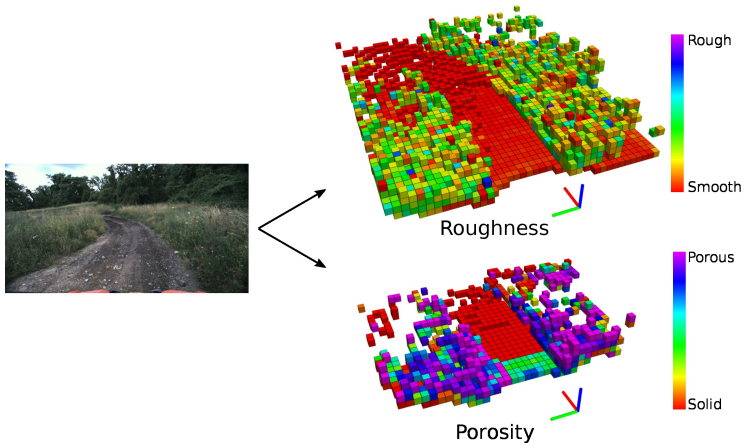


Figure 5.8: Visualization of roughness and porosity feature. The terrain area shows a low roughness and low porosity relative to the vegetation area. We omit empty voxels for visibility. The x, y and z axes are colored red, green and blue respectively.

subsection 5.2.2, in which the convolution, max-pooling, and deconvolution layers are lifted to 3D. However, for the sake of computational performance, we make some small changes. We replace the dilation and asymmetric layers with regular convolution layers, and we replace the deconvolution layers with nearest neighbor upsampling layers, followed by $3 \times 3 \times 3$ convolutions.

We want to predict the semantic classes of high vegetation and terrain, as these commonly appear in off-road scenes. Intuitively, we would expect high-vegetation surfaces to appear to be relatively "rough" compared to bare terrain surfaces; similarly, we would also expect high-vegetation surfaces to appear more "porous", or less dense, than bare terrain surfaces. Following this intuition, we encode features representing the roughness and porosity for each cell. These features are illustrated in Figure 5.8. This encoding differs to that of chapter 4, where each grid cell only has a single feature, the equivalent of porosity. Here, we opt for this representation to encode more spatial context in each cell, as we use slightly larger voxel sizes to compensate for the relative sparsity and noisiness of the point clouds from this platform.

Similarly to chapter 4, we use a voxel grid to describe the point cloud. For each voxel, indexed by (i, j, k), we obtain its roughness feature $R^{3D}_{i,j,k}$ by calculating the mean residual from a fitted plane to each point inside the voxel [154]:

$$R^{3D}_{i,j,k} = \frac{1}{N} \sum_{n=1}^{N} \frac{|Ax_n + By_n + Cz_n + D|}{\sqrt{A^2 + B^2 + C^2}} \tag{5.1}$$

where N is the number of points inside each voxel, x, y, z are the position of each point, and A, B, C, D are the fitted plane parameters for N points inside the voxel (*i.e.*, $Ax + By + Cy + D = 0$). We assign a constant negative roughness value of -0.1 to voxels with no points.

For the porosity feature $P^{3D}_{i,j,k}$, we use 3D ray tracing [1] to obtain the number of hits and pass-throughs for each grid voxel. Similarly to chapter 4, we model the porosity by updating the Beta parameters $\alpha^t_{i,j,k}$ and $\beta^t_{i,j,k}$ for the sequence of lidar measurements $\{z^t\}_{t=1}^{T}$ [120]:

$$\alpha^t_{i,j,k} = \alpha^{t-1}_{i,j,k} + z^t \tag{5.2}$$

$$\beta^t_{i,j,k} = \beta^{t-1}_{i,j,k} + (1 - z^t) \tag{5.3}$$

$$P^{3D}_{i,j,k} = \frac{\alpha^t_{i,j,k}}{\alpha^t_{i,j,k} + \beta^t_{i,j,k}} \tag{5.4}$$

where $\alpha^0_{i,j,k} = \beta^0_{i,j,k} = 1$ for all (i, j, k), $z^t = 1$ for hits, and $z^t = 0$ for pass-throughs.

5.2.4 Projection modules

The projection modules project the 3D features learned by the point cloud network onto 2D feature maps. They are followed by bottleneck modules to encourage the extraction of features from these feature maps. For this projection, we map each voxel's centroid position (x, y, z) with respect to the lidar onto the image plane (u, v) using the pinhole camera model:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R & | & t \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{5.5}$$

where f_x, f_y, c_x, c_y are the camera intrinsic parameters, R and t are the 3×3 rotation matrix and the 3×1 translation matrix from the camera to the lidar, respectively. We sample (x, y, z) for every voxel size from the original point cloud dimension (*e.g.*, $16 \times 48 \times 40$ in Figure 5.9). This is to address the sparsity of the projected data caused by the 3D max-pooling layers, which reduce the number of voxels in the 3D stream. We apply a z-buffer technique to account for pixels that have multiple lidar points projected onto the same pixel location. Then, we use nearest-neighbor interpolation to resize the projected feature maps to match the size of the image network layer that the projection module will be merged into (subsection 5.2.5).

We consider a fixed volume of 3D point clouds with regard to a lidar (section 5.2.6). Thus, the voxel locations and their corresponding projection locations in the image network are constant if the dimensions of a point cloud and an image are the same (*e.g.*, projection for stages 1 and 4). In practice, we precompute the indices of voxel locations and their corresponding pixel indices and use them inside the network.

5.2.5 Joint 2D-3D network

Figure 5.9 summarizes our multimodal 2D-3D network architecture: the point cloud network learns 3D features from the roughness and porosity volumetric inputs, the projection module propagates the 3D features to the image network, and the image network fuses the 3D features with the 2D features extracted from the image data. We apply the projection modules to the outputs of the initial stage and stages 1–5 to encourage the extraction of information at multiple scales [78].

5.2.6 Experiments

We evaluate our approach for the task of semantic segmentation for off-road navigation. We are particularly interested in the impact of using multimodal architectures on generalization. To this

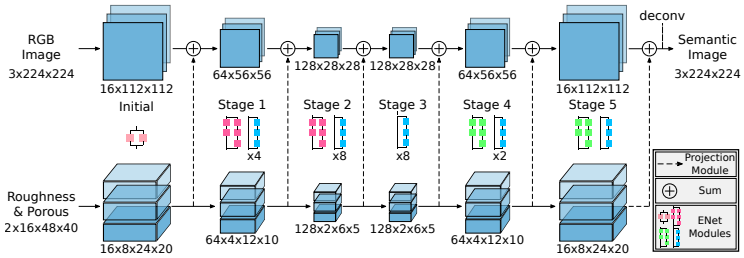


Figure 5.9: Our multimodal network architecture. The upper network path is the 2D image network, and the lower network path is the 3D point cloud network. They are connected via the projection modules. The ENet modules refer to the modules in Figure 5.7. The number below the bottleneck module indicates the number of times that the module is repeated.

end, we create a manually labeled dataset of image and lidar dataset pairs captured by our platform featuring variations in appearance caused by illumination, weather, and season. We compare our approach on this dataset against several baselines using different subsets of the image of point cloud data.

Dataset

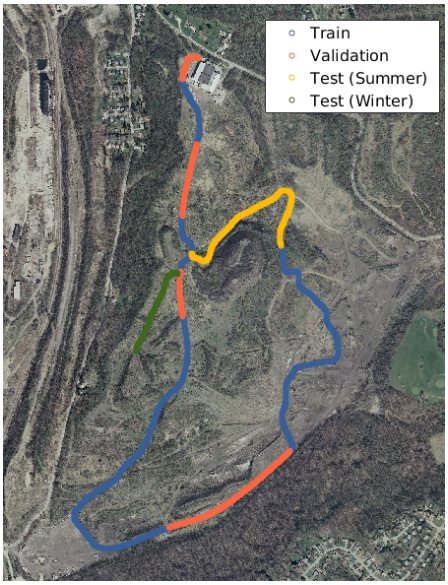


Figure 5.10: GPS coordinates of paths for different datasets

We collected our dataset using the same modified All-Terrain Vehicle (Figure 5.2) as in our prior experiments. To acquire a dataset with a large appearance variation, we collected our data on two separate dates: a sunny summer day in July 2016 (24 sessions) and a cloudy winter day in January 2017 (two sessions). Because the amount of winter data collected is considerably smaller, winter data is used only for testing. We divide the data as follows: 17 summer sessions for

training, four summer sessions for validation, and three summer sessions plus two winter sessions for testing.

For the K-fold cross-validation in section 5.2.6, we fix the test datasets, but randomly shuffle train/validation sessions. The geographic and split distribution for one of the K-fold cross validations is shown in Figure 5.10. We note that there is no overlap between the training, validation, and testing datasets. On average over the K folds, the training dataset has 7.2k image-point cloud pairs and the validation dataset has 1.7k pairs. The test data for summer has 1.3k pairs, and the test data for winter has 0.6k pairs.

Our ground-truth semantic labels consist of four classes: *High Vegetation*, *Rough Terrain*, *Smooth Terrain*, and *No Info*. To effectively label the ground-truth and minimize the human error, we first construct a registered point cloud by stitching point clouds over time (Figure 5.11a). We then manually label the registered point cloud in the point cloud space between the terrain and high-vegetation class (Figure 5.11b). We separately label another cloud with labels between the rough terrain and smooth terrain using equation Equation 5.1 (Figure 5.11c). We merge the two labeled point clouds into one cloud with three classes (Figure 5.11d). To get image labels, we project the final labeled point cloud onto the image plane. We consider voxels with no points and pixels with no lidar points projected as the *No info* class.

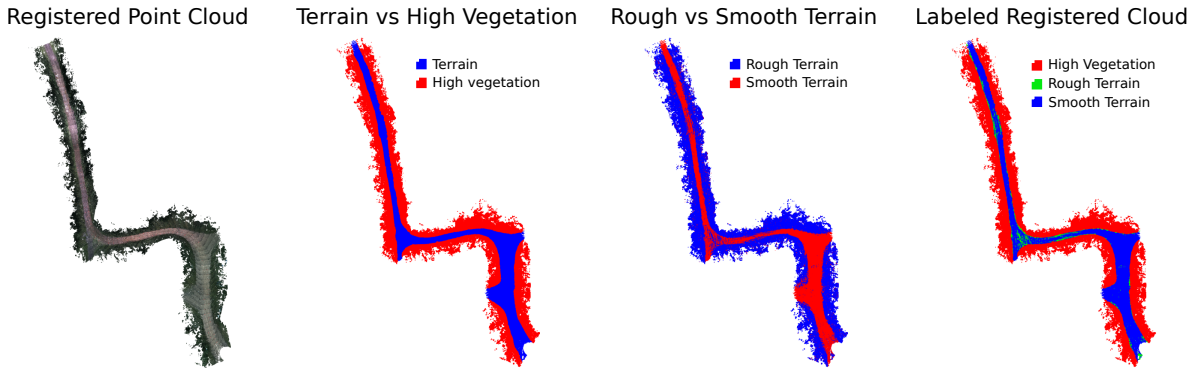


Figure 5.11: The point cloud ground-truth generation procedure. (a): Point clouds are first registered. (b): The terrain and high-vegetation class are labeled manually. (c): The rough and smooth terrain class are labeled automatically using equation 5.1. (d): Final labeled point cloud is acquired by merging labeled point cloud (b) and (c).

Baselines

We compare the performance of our method (*Ours-Proj*) against several baselines. The first baseline (Mode) classifies each pixel based on a pixelwise mode of the labels in the training dataset. Because the off-road data has a general structure of trail on the center and vegetation on the sides, this baseline is significantly better than chance. The second baseline, SegNet, is a popular encoder-decoder image segmentation network [5]. The third baseline, *Ours-Image*, is the image network of our multimodal network without the point cloud network and the projection modules. The last baseline, *Ours-RGBRP*, is the same as *Ours-Image*, but the input to the network is five

channels (RGB, Roughness, Porosity) obtained by projecting the point cloud network's inputs onto the image planes and treating them as additional channels, similarly to the color channels. The *Ours-RGBRP* baseline compares the effectiveness of the learning and propagation of 3D features against learning features only in 2D.

We explored different options for *Ours-Proj*, with projections only in the encoder (stages 1-3), projections only in the decoder (stages 4-5), and projections in both. Encoder-only projections and projections in both the encoder and decoder obtained nearly equivalent performance to each other and significantly better performance than the decoder-only projections, and we report results for encoder-only projections.

Training details

All input and label images are resized to 224×224 pixels. With respect to the lidar, we clip the point cloud to a fixed volume: $-3\,\mathrm{m}$ to $0.6\,\mathrm{m}$ for the z-axis (up direction), $3\,\mathrm{m}$ to $17.4\,\mathrm{m}$ for the x-axis (forward direction), and $-6\,\mathrm{m}$ to $6\,\mathrm{m}$ for the y-axis (left direction), where the axes are in the vehicle frame (see Figure 5.8).

The voxel size is $0.3\,\mathrm{m}$, so our voxel grid has size $12 \times 48 \times 40$ in the z-, x-, and y-axis, respectively. To reduce the GPU memory required for training *Ours-Proj*, we first train the point cloud network in isolation. We then remove the deconvolution and softmax layers in the point cloud network, attach the image network via the projection modules, and train the image network and projection modules with the point cloud network's weights frozen. Except for SegNet, all learning methods are based on Theano. For SegNet [5], we use its publicly available code. We train all learning methods from scratch and use the validation data to determine weights for testing.

Experimental results

Table 5.1: Quantitative results on Summer test (mean and standard deviation)

	Per-Class IoU				Average PR	
	Vege.	Rough	Smooth	No Info	Precision	Recall
Mode	.513 (.041)	.000 (.000)	.508 (.015)	.806 (.009)	.572 (.006)	.611 (.010)
SegNet	.816 (.008)	.182 (.007)	.670 (.019)	.828 (.010)	.741 (.003)	.767 (.008)
Ours-Image	.814 (.007)	**.183** (.008)	**.702** (.059)	.837 (.003)	.742 (.004)	.767 (.008)
Ours-RGBRP	.833 (.008)	.181 (.019)	.648 (.104)	.858 (.011)	**.747** (.007)	**.774** (.017)
Ours-Proj	**.839** (.005)	.179 (.014)	.655 (.072)	**.864** (.003)	**.747** (.006)	.772 (.015)

We report quantitative performance on the per-class Intersection over Union (IoU) and average precision-recall (PR) metrics (computed in 2D) in Table 5.1 and Table 5.2. The numbers correspond to the mean and standard deviation of the K-fold cross validations, where $K = 5$.

Unsurprisingly, the trivial Mode baseline is the worst performing method, specially for the *Rough* class, which has the least predictable spatial distribution. The performances between the unimodal networks (SegNet and *Ours-Image*) and the multimodal networks (*Ours-RGBRP* and

Table 5.2: Quantitative results on Winter test (mean and standard deviation)

	Per-Class IoU				Average PR	
	Vege.	Rough	Smooth	No Info	Precision	Recall
Mode	.453 (.010)	.000 (.000)	.712 (.012)	.855 (.003)	.589 (.002)	.609 (.004)
SegNet	.474 (.067)	.027 (.002)	.660 (.109)	.784 (.059)	.605 (.032)	.630 (.031)
Ours-Image	.498 (.018)	.017, (.004)	.595 (.120)	.862 (.009)	.623 (.008)	.622 (.020)
Ours-RGBRP	.582 (.035)	.036 (.008)	.692 (.107)	**.881** (.002)	.678 (.010)	.689 (.022)
Ours-Proj	**.620** (.012)	**.040** (.005)	**.790** (.061)	.875 (.002)	**.688** (.003)	**.705** (.010)

Ours-Proj) are comparable for summer sessions. However, the multimodal networks outperform the unimodal networks for winter sessions. For instance, *Ours-Proj* shows a 25% improvement in mean Intersection-over-Union (IoU) of the navigation-related semantic classes (*i.e.*, excluding the No-info class) relative to SegNet. Between *Ours-Proj* and *Ours-RGBRP*, *Ours-Proj* shows improved IoU and PR, in particular for the smooth terrain class. The results suggest that the learning and propagation of 3D features help the network learn more robust feature representations, which is also supported by qualitative examination Figure 5.12. Videos of the qualitative results can also be found at http://frc.ri.cmu.edu/~dk683/fsr17/fsr17.mp4.

The IoU scores for the rough terrain in the winter class are low, partially due to the relative rarity of this class in the winter data, mostly consisting of small and hard-to-detect regions. We note that the multimodal methods can have an advantage in predicting the no-info class because the ground-truth for the class is based on the lidar projection. However, the multimodal networks still show improved results for the navigation-related classes.

Feature visualization

To get a sense of how the 3D data is fused into the network, we plot selected feature activation maps from each of the projection layers in the network, using the configuration with projections in the encoder and decoder Figure 5.13. We find that there are activation patterns that are correlated with spatial structures, such as low horizontal surfaces (*e.g.*, terrain), vertical surfaces on both sides (*e.g.*, high vegetation), or more complex combinations of height, width, and depth. These are 3D spatial features that are hard to learn in the image domain, and intuitively, should be helpful for semantic inference.

5.2.7 Discussion

In this section we introduced a novel multimodal architecture for semantic segmentation from image and lidar data. This architecture shows promising quantitative and qualitative results that show it can learn to extract useful representations from each modality and fuse them to improve the accuracy of semantic segmentation.

However, for practical purposes, the gains in accuracy from our multimodal approach are relatively marginal in relation to its increased computational complexity, compared to image-only

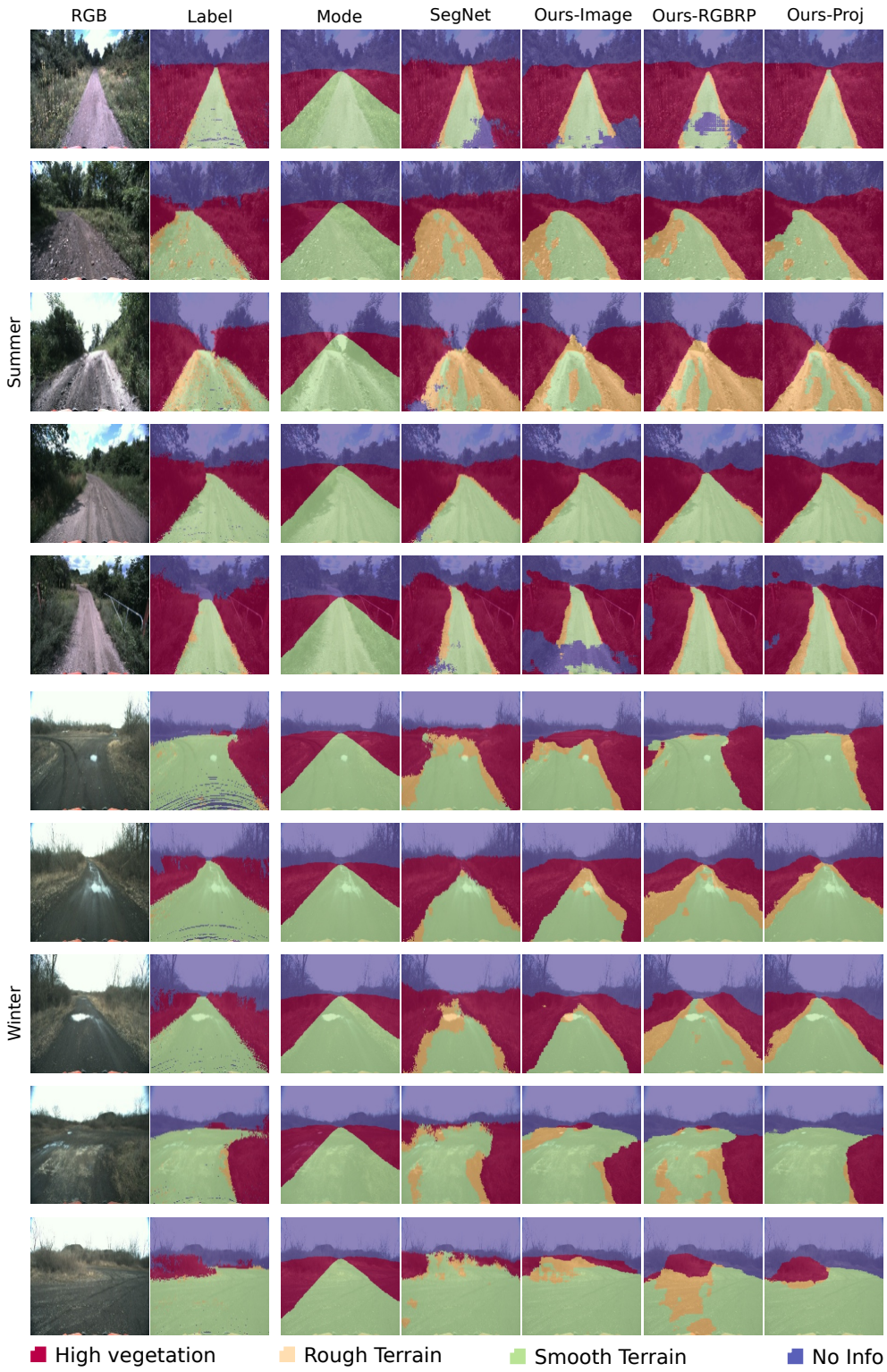


Figure 5.12: Predicted semantic segmentations from each method in summer scenes (top five rows) and winter scenes (bottom five rows).

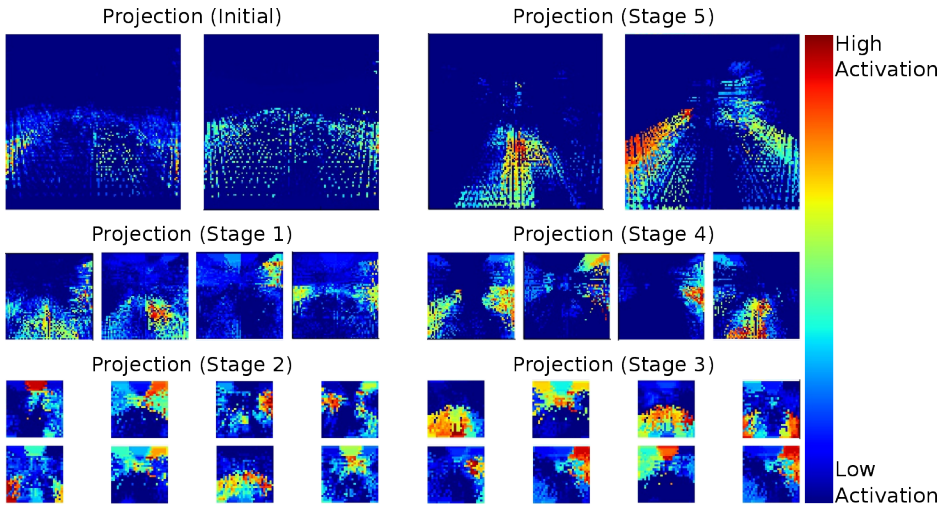


Figure 5.13: Feature map visualization for each projection module's output.

approaches. Currently, our implementation of the multimodal network has an inference time upwards of $600\,\mathrm{ms}$ for each image-point cloud pair, which is too slow for autonomous navigation at high speeds.

We leave the design and implementation of more efficient versions of this architecture as a topic for further research. Meanwhile, motivated by the accuracy and cost tradeoffs we observed for the different baselines in these experiments, we explore a simpler — and more efficient — to using multimodal data in the next section.

5.3 Decoupled multimodal approach to 2.5D semantic mapping

In the experiments from section 5.2, we observed that for the purposes of 2D semantic segmentation, image-only approaches achieve only slightly lower accuracies than the multimodal input approaches, especially when the training and testing data are drawn from the same season. This suggests that an image-only approach can perform sufficiently well for autonomous navigation purposes in terms of accuracy, particularly when trained on a representative dataset. At the same time, we observed image-only CNN inference is computationally more efficient by a large factor, suggesting that a semantic mapping using 2D-only semantic inference could also have sufficiently low latency for autonomous navigation.

Based on these observations, in this section we develop a system for 2.5D semantic mapping with a *decoupled* approach to inference of the semantic and spatial properties that we represent in our maps. Semantic information is predicted solely from the image data, whereas the spatial information is built from the lidar data. This choice exploits the relative strengths of each sensor modality and simplifies the development of modules meeting the required accuracy and efficiency for each task. Moreover, this decoupling also simplifies the task of data acquisition, as only image data (as opposed to image and lidar pairs) is required. Our resulting contribution is a simple yet effective system leveraging a custom CNN architecture, based on Fully Convolutional

Networks [113], and a 2.5D vehicle-centered semantic grid map that fuses the geometric and semantic measurements. We show the effectiveness of the semantic segmentation CNN in offline benchmarks, including a new dataset gathered for this work. As a proof of concept, we implement a reactive planner that uses the semantic map to successfully navigate through challenging off-road terrain.

5.3.1 Overview

To address these challenges, we use our two main sensor streams — the camera images and lidar point clouds — in a way that plays to the strengths of each. This system architecture is outlined in Figure 5.14.

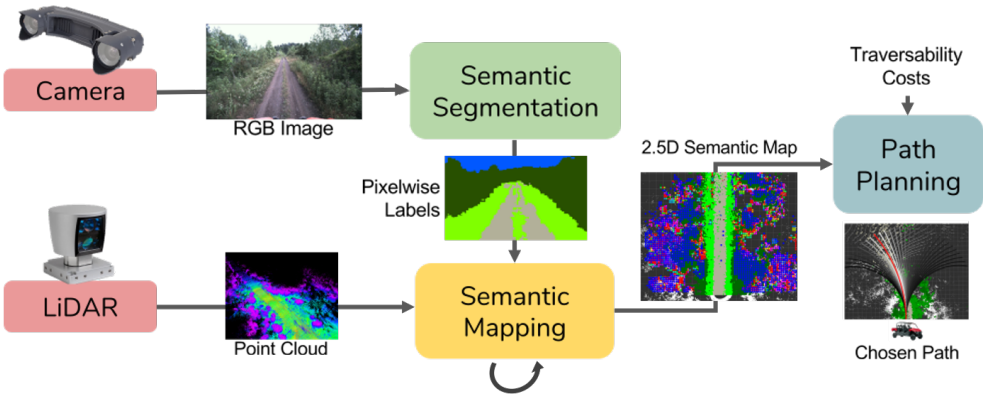


Figure 5.14: Outline of decoupled semantic mapping and trail following system

Monocular camera imagery captures high-resolution color and texture information that is highly informative — at least to the human eye — as to the traversability and semantic class of visible surfaces. We use this sensor stream to create pixelwise semantic segmentations of each image using a custom CNN.

On the other hand, image data lacks direct depth information that is critical for navigation planners that use metric spatial maps. The lidar point clouds of this platform, while relatively sparse in their surface coverage, directly convey 3D information useful for navigation. We accumulate and summarize this information over time with an 2.5D elevation map.

To fuse the output of both modules, the semantic mapping module projects the pixelwise prediction images from the semantic segmentation module onto the elevation map, which also smoothes these predictions over time. The result is a vehicle-centered 2.5D elevation map encoding continuously updated estimates of relevant geometric and semantic information for off-road navigation.

The map can be used for semantically-aware path planning. We developed a reactive receding-horizon path planner that assigns a traversal cost to each semantic class and continuously chooses a path to minimize the cost. The whole system runs at 10 Hz, a rate dictated by the speed at which the semantic segmentation module processes images.

5.3.2 Semantic segmentation architectures

The goal of 2D semantic segmentation is to assign one of K predefined classes to each pixel in an image. As we discussed in chapter 2, Convolutional Neural Networks (CNNs) are currently the *de facto* standard approach to learning-based classification in computer vision. For the task of semantic segmentation, the state of the art has been recently set by *Fully Convolutional Networks* (FCNs) [113].

The key idea in FCNs is to replace the fully connected layers of typical CNNs with convolutions. After this "convolutionalization" process, the output of a FCN is a feature map encoding a low-resolution semantic segmentation of the input, which can be computed in a single feed-forward pass.

However, due to pooling, the results in low-resolution outputs; to reverse this, transposed convolution layers are added to upsample the output. In order to preserve high-frequency detail, skip layers connecting early layers to upsampled feature maps are added. Encoder-Decoder architectures [132, 5], of which UpNet [177] is an example, are similar, but omit skip layers.

We found the state of the art architectures to be too computationally intensive four our needs. Thus, we designed more lightweight alternatives, drawing inspiration from recent architectures in single-shot object detection [144].

We investigate two custom architectures. The first, `cnns-fcn`, is based on our "convolutionalization" of the VGG-CNNs from Chatfield et al. [28]. It has a 227×227 input size and a 109×109 output size. While it is common practice for FCNs to have equal input and output sizes, we chose to output a smaller size by omitting the last upsampling layer to reduce computational costs. The second, `dark-fcn`, is based on our convolutionalization of the Darknet architecture [144], which in turn is a slimmed down version of VGG16 [161]. For `dark-fcn`, both the input and output are 300×300; for this network, we found it was feasible to use an output size equal to the input size. The specific number of 300×300 is also the resolution used by UpNet. Despite the higher output resolution, `dark-fcn` is faster than `cnns-fcn`; on a mobile GT980M GPU, we measured its its latency as $21\,\mathrm{ms}$, compared to $37\,\mathrm{ms}$ for `cnns-fcn`. The authors of UpNet [177] describe a $50\,\mathrm{ms}$ with Caffe on a GTX Titan X, which in our experience has similar speeds to the GT980M. This leads us to believe our models should be faster, or at least comparable to UpNet. Figure 5.15 shows both of our architectures.

5.3.3 2.5D semantic mapping

The output of the semantic segmentation step is in 2D image space, but it is far more natural for vehicles to plan in a metric space. For our system, we adopt a 2.5D or elevation map representation, encoded in a grid with per-cell height and label estimates. This representation is memory-efficient and suffices for most environments we encounter, but is likely to have issues with overhanging trees or tunnels.

To keep an up-to-date elevation map of the vehicle's surroundings, we use a scrolling grid data structure [90]. This structure is a generalization of ring-buffers to two dimensions, and its main feature is that it can be shifted (translated) without copying its data, and instead updating the variables indicating its limits. This is a speed optimization; logically, the grid behaves like a finite 2D array centered around the vehicle, with each grid cell containing various properties about the

Name	Units	Size	Stride	Input
conv1	96	7	2	
norm1				
pool1		2	2	
conv2	256	5	1	
pool2		2	2	
conv3	512	3	1	
conv4	512	3	1	
conv5	512	3	1	
pool5		3	3	
convfc6	4096	6	1	
convfc7	4096	1	1	
ninfc7	6			
up1	6	4	2	
conv5nin	6			conv5
fuse1				up1 + conv5nin
up2	6	4	2	
pool1nin	6			pool1
fuse2				up2 + pool1nin
up3	6	5	3	
conv1nin	6			conv1
fuse3				up3 + conv1nin

cnns-fcn

Name	Units	Size	Stride	Input
conv01	32	3	1	
pool01		2	2	
conv02	64	3	1	
pool02		2	2	
conv03	128	3	1	
conv04	64			
conv05	128	3	1	
pool03		2	2	
conv06	256	3	1	
conv07	128			
conv08	256	3	1	
pool04		2	2	
conv09	512	3	1	
conv10	256			
conv11	512	3	1	
conv12	256			
conv13	512	3	1	
pool05		2	2	
conv14	1024	3	1	
conv15	512			
...				

Name	Units	Size	Stride	Input
conv16	1024	3	1	
conv17	512			
conv18	1024	3	1	
conv18nin	64			
up1	64	4	2	
pool04nin	64			pool04
fuse1				up1 + pool04nin
up2	64	4	2	
pool03nin	64			pool03
fuse2				up2 + pool03nin
up3	64	4	2	
pool02nin	64			pool02
fuse3				up3 + pool02nin
up4	64	4	2	
up4nin	8			up4

dark-fcn

Figure 5.15: Architecture of our 2D semantic segmentation networks. *conv* denotes a convolutional layer; *pool*, a pooling layer; layers ending in *nin* are 1×1 convolutional layers; *fuse* is an elementwise sum layer; *norm* is a local response normalization layer. The input is assumed to be the layer above, unless otherwise specified. For convolutional layers, *size* is the kernel size; for pooling layers, it is the pooling receptive field. Note that for `dark-fcn` we split the table in two columns due to space constraints.

terrain. In our system, the grid cells are $0.25 \times 0.25\,\mathrm{m}^2$ each, and the map has 400×400 cells. Each grid cell maintains a running estimate of the minimum and maximum height in that grid cell, computed by using occupancy and free-space constraints derived from lidar rays, similar to [75, 191]. For each point in the point cloud, we raytrace on our grid using Bresenham's algorithm in 3D; cells that are passed through, and above, are considered empty, and cells where the beam stops, and below, are considered occupied.

The semantic map also integrates semantic measurements, as its name indicates. To project the output of the 2D semantic segmentation into the elevation map representation, we use the intrinsic parameters of the camera and its relative pose with respect to the lidar, both of which are obtained by a prior calibration step. As shown in Figure 5.16, we cast a ray for each pixel in the image and find its intersection with the height map using a 3D line rasterization method [1].

For added robustness, we fuse measurements over time. To this end, we adopt a scheme inspired by the sequential filtering process of occupancy maps [122]. Using the log-softmax outputs for each pixel, we maintain a running sum of the log-odds of the K classes projected to each grid cell. While this soft multiclass representation could be used directly, for planning purposes, we use the argmax over the K classes as our current best estimate of the semantic class for each grid cell. This representation assumes a single class per cell, which may be a limitation in certain environments.

Finally, we accumulate the log posterior probabilities of each class in the corresponding cells according to:

$$l_k^{(t+1)}(g_x, g_y) = l_k^{(t)}(g_x, g_y) + \log P(Y = k | i, j, \mathbf{I}^{(1:t)}) \tag{5.6}$$

where $\mathbf{I}^{(1:t)}$ is the sequence of RGB images up to frame t, and $logP(Y = k|i, j, \mathbf{I}^{(1:t)})$ is the log

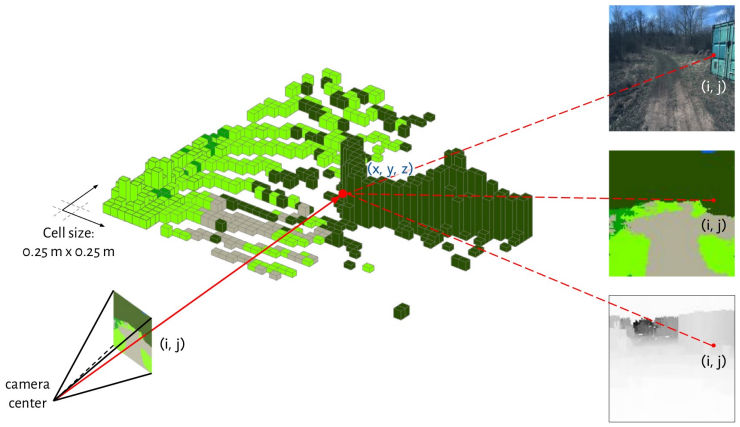


Figure 5.16: Projecting the 2D semantic segmentation on the 2.5D map.

posterior probability of class k at image coordinate (i, j).

An example cumulative output of the semantic map in a live field run is shown in Figure 5.17.

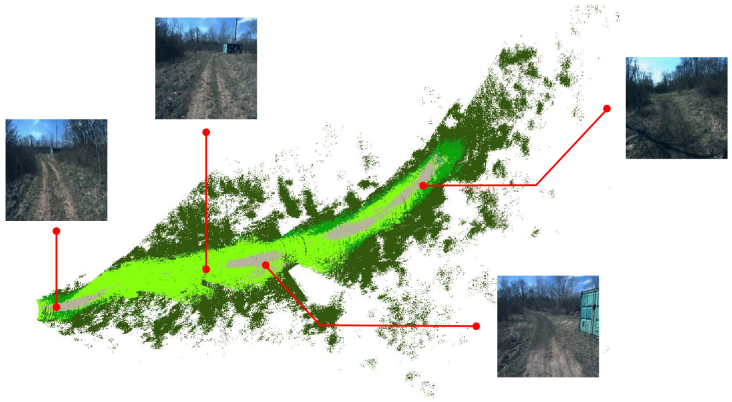


Figure 5.17: Example output of semantic map in a live field run

5.3.4 Reactive path planning

To demonstrate autonomous operation, we implement a simple receding horizon path planner. The planner has a library of 30 trajectories corresponding to yaw rates of $-15\,^\circ/\mathrm{s}$ to $15\,^\circ/\mathrm{s}$, discretized at $1\,^\circ/\mathrm{s}$, with a constant velocity of $9\,\mathrm{km/h}$; see Figure 5.18a.

Each time the map is updated, which happens at $10\,\mathrm{Hz}$, a trajectory is chosen from the library. The choice of trajectory maximizes a reward function derived from the semantic map as follows. Cells labelled as "smooth" or "rough" trail have a reward of 1, and cells labeled as "grass" have a reward of 0.1. All other classes have zero reward. The total reward of a trajectory is the sum of rewards over a $20\,\mathrm{m}$ trajectory length, originating from the vehicle. To account for vehicle width, we slightly modify this calculation, as shown in Figure 5.18b.

The advantage of this planner is that in its extreme simplicity, its performance depends largely on the output of our semantic mapping, with no interference from other factors that will be present

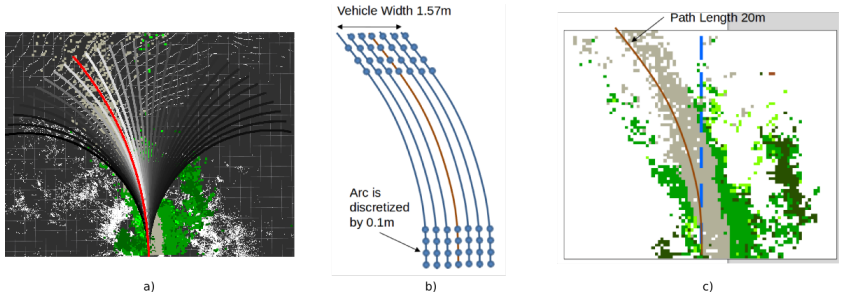


Figure 5.18: The reactive path planner. (a): Library of candidate paths, overlaid on top of the semantic map. Red indicates feasible paths. (b): Illustration of how we account for vehicle width. For each trajectory, we compute the cost (or reward) over seven shifted versions of the trajectory, covering the vehicle footprint. (c): An example chosen trajectory, chosen according to the traversability score of the semantic classes it covers.

in a more complex, multi-layered system. However, our system has also been used as an additional input to a more deliberative planner, for which the main representation is a geometric map built with lidar. In this planner, our semantic predictions were used primarily to allow the ATV to traverse grass and drive through narrow trails surrounded by vegetation.

5.3.5 Experiments

We evaluate our system in two ways. First, we run offline benchmarks of the semantic segmentation module on two datasets. Second, we demonstrate the whole system operating autonomously in live field experiments.

Offline Benchmarks

In order to evaluate our semantic segmentation module we use two datasets, the DeepScene dataset from Valada et al. [177] and our own, the Yamaha-CMU Off-Road dataset.

DeepScene dataset This dataset consists of 233 training images and 139 validation images of off-road imagery densely labeled with six semantic categories: void, road, grass, vegetation, tree, sky, and obstacle. While this dataset shows some interesting variety in appearance due to the time of day, it is fairly small and seems to lack diversity in terms of weather and location. A key feature of this dataset is various modalities (depth, NIR), but we do not currently make use of them.

Yamaha-CMU Off-Road dataset To train and evaluate our method, we have collected our own dataset, which we call Yamaha-CMU-Off-Road, or YCOR. It consists of 1076 images collected in four different locations in Western Pennsylvania and Ohio (Figure 5.20), spanning three different seasons (Figure 5.19). The dataset was labelled using a polygon-based interface with eight classes: sky, rough trail, smooth trail, traversable grass, high vegetation, non-traversable low vegetation, obstacle. The polygon labels were postprocessed using a Dense CRF [97] to densify the labels;

the output of the CRF was manually inspected, and in some cases corrected, to ensure no wrong labels were created.

We believe our dataset is more diverse and challenging than DeepScene. In Figure 5.20, we show the mean RGB image and pixelwise labelmode of each dataset. The DeepScene dataset shows a left-right bias and more predictable structure than ours; if we used the pixelwise mode as a baseline classifier, we would obtain 0.30 pixelwise error-rate in DeepScene, but 0.51 in ours. However, we acknowledge that compared to recent efforts, both datasets are relatively small; cf. CityScapes [38], with 25000 labeled images.

Figure 5.19: Montage of frames from the YCOR dataset.

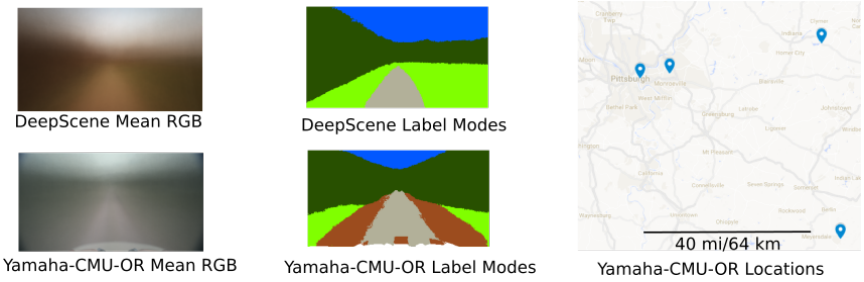


Figure 5.20: First two columns: A visual comparison of dataset statistics. We show the mean RGB frame and the pixelwise mode for the labeled frames in the training sets of each dataset used. Last column: a map with locations where YCOR was collected.

Our current split has 931 training images and 145 validation images. This split was generated randomly, ensuring there was no overlap in the data collection sessions between images in the training and validation split. However, there is overlap in the locations used, given the limited number of trails available to use for field testing.

Quantitative Results We evaluated our models on the two datasets. In each case, we train our models from scratch on the predefined training set until convergence with SGD, dividing

by the initial learning rate (0.0001) by a factor of 10 three times. We use a standard pixelwise cross-entropy loss with a small L_2 regularization factor (0.0005). Training takes around two days on a GT980Ti GPU. We use crop, rotation and color augmentations at training time, and none at test time. We use per-class intersection over union (IoU) as the evaluation metric, the most common metric for semantic segmentation.

Table 5.3 shows results for DeepScene and Table 5.4 shows results for YCOR. In both, we include a variant of the `dark-fcn` model with 448×448 resolution, in addition to the standard 300×300. We report the numbers from their paper [177], where we denote by frequency-weighted IoU (fw-IoU) what they denote as IoU, and add mean IoU (mIoU), calculated by ourselves. As we can see, both our models perform comparably, with `dark-fcn` having a slight advantage. In the DeepScene dataset, we can also compare the two models with the RGB UpNet. We see that our models have a slight edge in fw-IoU, though they display dramatically worse performance for obstacles, which severely skews the mIoU metric. We note that the number of obstacle pixels in the dataset is three orders of magnitude less than for the other classes, so the network tends to ignore it. Nonetheless, it is an important class, and further investigation should be conducted into improving accuracy for this class. Finally, we see that increasing the input resolution gives a slight boost in performance.

Table 5.3: Per-class, mean IoU and frequency-weighted IoU of UpNet (RGB) and our models in DeepScene dataset. The first three rows use a 300×300 image size, as in UpNet; the last row uses 448×448.

	road	grass	veg./tree	sky	obstacle	mIoU	fw-IoU
Upnet-RGB [177]	85.03	**86.78**	**90.90**	90.39	**45.31**	**79.68**	85.30
`cnns-fcn`	85.95	85.34	87.38	90.53	1.84	58.51	87.47
`dark-fcn`	**88.03**	86.65	89.21	**93.17**	5.03	60.35	**89.41**
`dark-fcn-448`	88.80	87.41	89.46	93.35	4.61	60.61	89.85

Table 5.4: Per-class, mean IoU, and frequency-weighed IoU of our models in the YCOR dataset.

	smooth	grass	rough	puddle	obstacle	low veg.	high veg.	sky	mIoU	fw-IoU
`cnns-fcn`	**46.70**	64.03	38.29	0.0	32.74	24.32	79.15	88.01	46.66	61.31
`dark-fcn`	46.24	**71.25**	**41.35**	0.0	**29.74**	**28.17**	**80.15**	**91.45**	**48.54**	**63.62**
`dark-fcn-448`	52.46	72.11	39.61	0.0	35.56	24.61	82.51	92.69	49.82	65.18

Qualitative Results We show some qualitative labellings of the `cnns-fcn` architecture for each dataset in Figure 5.21. As can be seen, the results are generally quite accurate. For the YCOR, most of the confusions come from smooth vs. rough trail, a distinction that is hard for humans to make consistently.

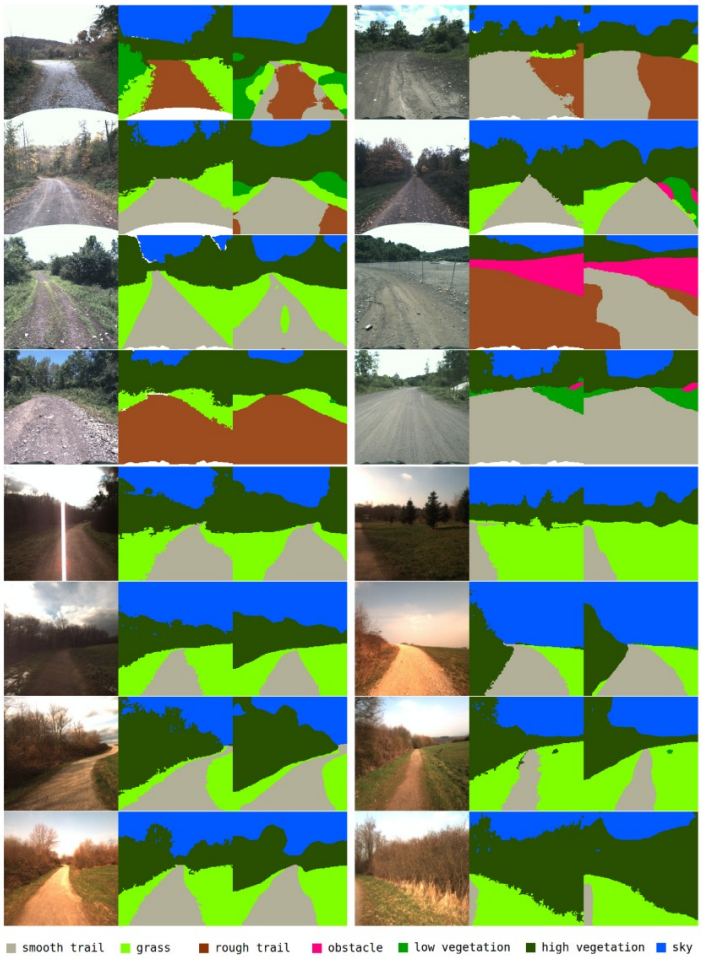


Figure 5.21: Montage of predictions from `cnns-fcn` in the YCOR dataset (top four rows) and DeepScene (bottom four rows). In each case, we show three images: input, ground truth labels, and predicted labels.

Field Experiments

We perform field testing with the Erik ATV (Figure 5.2) introduced in section 5.1, using its 64-line Velodyne lidar for point cloud acquisition and RGB images from its frontal RGB stereo camera. Given the higher quality of the point clouds we obtain from the lidar sensor, we currently do not use depth information from the stereo camera.

All computation for the semantic mapping and planning modules is performed onboard with two commercial off-the-shelf laptops, connected via high-speed ethernet. Both laptops use Intel i7 CPUs and the laptop for semantic mapping includes a 6GB NVIDIA GT980M GPU, used to achieve real-time execution of the CNN classifier.

All computers run Ubuntu Linux. The different system modules run concurrently as ROS nodes and communicate through ROS messages. The nodes are implemented in C++ and Python, using CUDA (generated via the Theano library [8]) to make effective use of the GPU.

We performed self-driving experiments in March and July 2017 in various locations around our testing site near Pittsburgh, PA. Despite the simplicity of our planner, the vehicle managed to successfully traverse various trails that were too challenging for a previous lidar-only system. These include locations with puddles, grass in the middle of the trail, and narrow trails. Figure 5.22 shows the vehicle in autonomous operation. Videos can be found in `https://youtube.com/playlist?list=PLeg9sULe3rSlL211MEvy29wKeeDj-Nt1e`.

On the other hand, we observed some limitations of our current system; Figure 5.23 shows three typical failure cases. We believe that many these limitations can be largely mitigated by using a more sophisticated planning system; for example, the failures in Figure 5.23a and Figure 5.23b can be avoided by incorporating more spatial and temporal context in planning (*e.g.*, by path tracking and waypoint following), as well as a more flexible action space (*e.g.*, allowing the vehicle to slow down and reverse). The failure from Figure 5.23c can be avoided by reasoning about partially observed surfaces, as opposed to optimistically assuming unknown space is free. There is a vast literature of more deliberate planning methods that deal with these and other issues, most of which can take advantage of semantic map representations.

In fact, our semantic mapping system was also integrated into the proprietary mesh-based framework mentioned in section 5.1. In addition to its mesh-based terrain modeling, this system also uses more sophisticated path planning and tracking methods to achieve more stable trail following at higher speeds. Integrating our semantic maps enabled this system to traverse vegetated and narrow trails by relaxing its obstacle avoidance criteria in regions classified as vegetation. Figure 5.22b shows the ATV being driven by the mesh-based framework, in conjunction with our semantic maps, to successfully traverse the same trail that was previously considered untraversable in Figure 5.1.

We also saw failures caused by our semantic classification system. For example, it sometimes failed to detect sparse grass alongside the trail, resulting in the vehicle veering off-trail. On one occasion, it also confused a large non-traversable bush with traversable grass, forcing us to manually intervene. We also found our system to be unreliable for puddle detection, despite this being one of our cases of interest at the outset of the work in this chapter. Puddles are challenging due to the their specular nature; this causes their appearance to be highly dynamic and similar to other classes in their surroundings, at least in terms of local patch regions.

(a) Semantic predictions in image space and chosen paths during autonomous run using our reactive planner.

(b) Aerial views and top-down display of the semantic map traversing a narrow trail, courtesy of Mesh Robotics LLC

Figure 5.22: Action shots of our semantic mapping pipeline enabling traversal of vegetated, narrow trails. Videos can be found in https://youtube.com/playlist?list=PLeg9sULe3rSlL211MEvy29wKeeDj-Nt1e.

(a) Wrong turn into a cul-de-sac

(b) Swerving to avoid puddles

(c) Turning into previously unseen obstacles

Figure 5.23: Three failure cases in our field tests. In fig. (a), the vehicle is faced with a fork in the trail, and due to its short planning horizon, chooses the right path, which leads to a cul-de-sac. In fig. (b), the vehicle makes an aggressive turn to avoid a small puddle, after which it struggles to stay on the trail. In fig. (c), after a left turn, the vehicle comes too close to obstacles that were previously unseen due to occlusion.

Timing We observed an average latency of $35\,\mathrm{ms}$ for the semantic segmentation of each image, $60\,\mathrm{ms}$ for the 2.5D label projection step, and less than $5\,\mathrm{ms}$ for trajectory selection. As these steps occur sequentially, the system has an update rate of approximately $10\,\mathrm{Hz}$, and the bottleneck of the system is in the semantic mapping steps. While $10\,\mathrm{Hz}$ is sufficient for operation at speeds of up to $16\,\mathrm{km/h}$ with our reactive planner, higher update rates are desirable for faster speeds. There are many opportunities for performance optimization in our current system; for example, the label projection step could be parallelized and ported to the GPU. Likewise, the inference time of the semantic segmentation step could likely be lowered with platform-specific optimizations.

5.4 Summary

We proposed two systems to fully take advantage of camera and range sensors for semantic mapping.

In the first, we create a joint 2D-3D architecture for semantic inference, that fuses the 2D information from images and 3D information from point clouds in a learned fashion, and show the benefits of this joint architecture for semantic segmentation from lidar and image data.

In the second, we adopt a decoupled architecture using a 2D CNN for semantic inference from images and a height mapping module to build maps from lidar, and fuse both to create a semantic map by geometric projection.

There are multiple directions on which this work can be improved, some of which have been investigated by research published after the work conducted in this chapter. For example, one of the more obvious limitations of our 2D-3D CNN architecture is that it only propagates information from the 3D CNN to the 2D CNN; intuitively, propagating information in the opposite direction can be useful to improve the quality of the learned 3D features, specially in tasks involving a 3D output. Dai and Nießner [42] propose an approach that implements this idea and apply it to the task of 3D semantic segmentation.

The computational cost of simultaneously using 2D and 3D CNNs is also an issue which limits the applicability of joint multimodal approaches. Recent approaches have adopted different formulations for joint analysis of point cloud and image data to achieve higher computational performance. For example, Chen et al. [32] adopts a bird-eye view approach that allows the use of more efficient 2D convolutions for the task of 3D object detection. Another interesting example is Qi et al. [140], which use graph neural networks to create a sparser, dynamic representation of the 2D and 3D information in RGBD data for the task of semantic segmentation.

Chapter 6

Long-range semantic mapping for semantic exploration

> We live boxed-up lives. Our ancestors were always looking around. They surveyed the environment, for they needed to know where they were and what there was in all directions.
>
> J. J. Gibson
> *The Ecological Approach to Visual Perception*

In chapter 4 we developed a system for range-only semantic classification; in chapter 5 we propose a multimodal system for semantic mapping using range and image sensors. However, relying on lidar sensors is not ideal in all applications. In the pursuit of a (literally) lightweight perception system for Micro Aerial Vehicles, this chapter explores a primarily vision-only semantic mapping system.

6.1 Looking forward

Micro-Aerial Vehicles (MAVs) can quickly and inexpensively gather information with cameras, lidar, and various other sensors, due to their agility. This makes them invaluable for applications such as search and rescue [40, 56], infrastructure inspection [187, 10], surveillance [146], crop and wildlife monitoring [3], etc.

A common trend in these applications is that not all possible locations are of equal value; we are usually more interested in gathering information about specific targets, such as survivors, vehicles, animals, etc. Often, we do not know in advance the location of these targets, making it necessary to locate them before more detailed inspection. For example, in a disaster scenario we might be interested in searching for survivors and then approach them to capture high resolution images. Equipped with cameras, MAVs are able to switch from viewing large spaces at a distance to flying in closely to obtain more accurate information. This capability of gaining information at different scales makes MAVs excellent platforms for the aforementioned applications. We will

refer to the overall task of searching and gathering data for a semantic class of interest as scouting. Our goal is to create a system to enable MAVs to perform effective general-purpose autonomous scouting.

Towards this goal, we study the more concrete scenario depicted in Figure 6.1. In this scenario, we wish to find any cars within a predefined region and capture high-resolution imagery (*e.g.*, for 3D reconstruction). The location, number, and appearance of cars, if any, are not known *a priori*. We have a limited power budget, equivalent to around ten minutes of flight time.

Figure 6.1: Overview of the scouting task. 1) The vehicle is tasked with mapping a semantic class (here, *car*) with unknown prior location(s). 2) Using the system described in this chapter, the vehicle uses visual and positional information to create a 2.5D semantic map on the fly. 3) Using the map, the vehicle flies up to the desired class and acquires high-quality imagery, useful for tasks such as 3D reconstruction.

In support of this goal, this work in this chapter proposes a novel semantic mapping system to estimate the presence and metric location of the semantic classes of interest (*e.g.*, cars) in its surroundings, so a separate planning system (beyond the scope of this chapter) can create information-gathering plans. The map is continuously updated on-the-fly from forward-looking camera imagery and global state estimation, using only on-board computing.

The choice to use a nearly forward-facing (tilted downwards at 15°) camera is worth noting. We use this arrangement, as opposed to a downward-facing camera, in order to be able to perceive longer

ranges quickly without needing to fly long distances or extremely high altitudes. Unfortunately, this also makes recognition and reconstruction more difficult. While having both forward-facing and downwards-facing cameras is an attractive option, there is a tradeoff in terms of the added weight and complexity of the system, and in our system we have opted for simplicity.

There are several challenges in the design of such a system. First, the recognition of semantic categories from visual data is a non-trivial task; in this case the difficulty is compounded by the fact that in MAVs with forward-facing cameras, objects will have a highly variable appearance as they are captured from different heights and angles. Second, it is challenging to reconstruct 3D metric maps from monocular imagery, specially for distant objects and using image sequences captured from arbitrary camera motion patterns, as opposed to motion patterns deliberately crafted to aid reconstruction. Finally, to be useful, our system must operate in online and in real time, using the relatively constrained on-board computing on our vehicle.

To face these challenges, we make the following contributions:

First, we design a custom Deep Learning architecture for 2D semantic segmentation that achieves a good accuracy/speed trade-off for our application. Our starting point is recent convolutional architectures [113, 145] but we empirically make various modifications to optimize for our scenario. To train this network, we assembled and labelled a new dataset consisting of oblique aerial imagery gathered from publicly available videos, as well as our own field data.

Second, we propose a new 2.5D mapping system to efficiently estimate the location of the semantic classes found by the semantic segmentation system. Instead of solving the full 3D reconstruction problem, we assume we have access to a Digital Elevation Map (DEM) of the region, and we project the 2D measurements on this map, while exploiting for available semantic knowledge. DEMs are freely available for many places in the world, including most of the United States.The mapper fuses measurements over time, making use of knowledge regarding typical heights of objects to improve its accuracy.

We evaluate each part of our system and show the integrated system autonomously completing a data gathering mission in the field.

6.2 Related work

Semantic segmentation

Semantic segmentation of RGB imagery is a heavily studied topic in computer vision. As for most classification tasks in computer vision, the state-of-the-art has been considerably advanced by Deep Learning.

In particular, Fully Convolutional Networks [113] proposed a highly effective and discriminative model adapting state-of-the-art models for object recognition [161] for the task of pixelwise prediction with purely feed-forward computation.

Similar models [51, 132, 5] were proposed at approximately the same time. Since then, most work has focused on optimizing accuracy (*e.g.*[188]), but relatively little attention has been paid to computational cost, and in particular, per-image speed. Recent exceptions include ENet [137] and SceneNet [177], which we intend to evaluate in the future.

Spatial mapping

Reconstruction of geometric maps from visual and (optionally) inertial information is a well studied topic in the Simultaneous Localization and Mapping (SLAM) [20] and Structure from Motion (SfM) literatures.

Algorithmic and computational advances have made it feasible to employ these systems in robotics for real-time decision making; recent relevant examples include Faessler et al. [57] and Forster et al. [61], which use recent work in visual SLAM to create elevation maps from downward-facing cameras on UAVs.

While this work shows impressive results, it is not applied to frontal-facing cameras, a considerably harder problem, given the relatively small (or non-existent) parallax induced by camera motion in this scenario, specially for distant objects.

Semantic mapping

Some form of semantic mapping in robotics frequently arises in robotic systems using both semantic and spatial information to navigate; see [96] for a review and taxonomy.

Sengupta et al. [157] present an influential system using images and depth to create 3D segmentations for street-level imagery. A more recent, similar approach is [179].

Brostow, Fauqueur, and Cipolla [16], and more recently, [100] use monocular imagery for 3D reconstruction. In the aerial vehicle domain, Bryson et al. [17] is an interesting system using a fixed-wing platform to monitor vegetation in farm lands. Khan, Masselli, and Zell [92] classify terrain images from low-altitude imagery.

An impressive recent work is Delmerico et al. [44], which performs terrain classification with a UAV to support search search and rescue missions. Another impressive work is Desaraju et al. [47], which uses vision to find landing zones. Most of these works use top-down imagery, and it is unclear how their results generalize to oblique imagery. In addition, computation is performed off-board. An exception using frontal imagery is Giusti et al. [68], but in this case the network only predicts discrete directions of travel, not dense labelings.

In summary, to our knowledge on-line semantic mapping, on-board an MAV is still an open problem when using oblique monocular imagery. The main challenge is posed by the sensing geometry that results in pixel measurements that are dependent on-each other and restricted computational resources.

6.3 Approach

The goal of the semantic mapping system is to inform the planning system about the presence and approximate location of the classes of interest in its surroundings, so it can create information-gathering plans. It does so by means of a *semantic map*, a metric map that is annotated with localized predictions regarding semantic classes [96].

Thus, to be useful, the system must operate online and in real time, in order to keep the map updated as new sensor data is acquired. Additionally, it must also be capable of recognizing and localizing distant ($40\,\mathrm{m}$ to $200\,\mathrm{m}$) objects, as its function is primarily to help the vehicle decide where to go, and secondarily to describe where it has been.

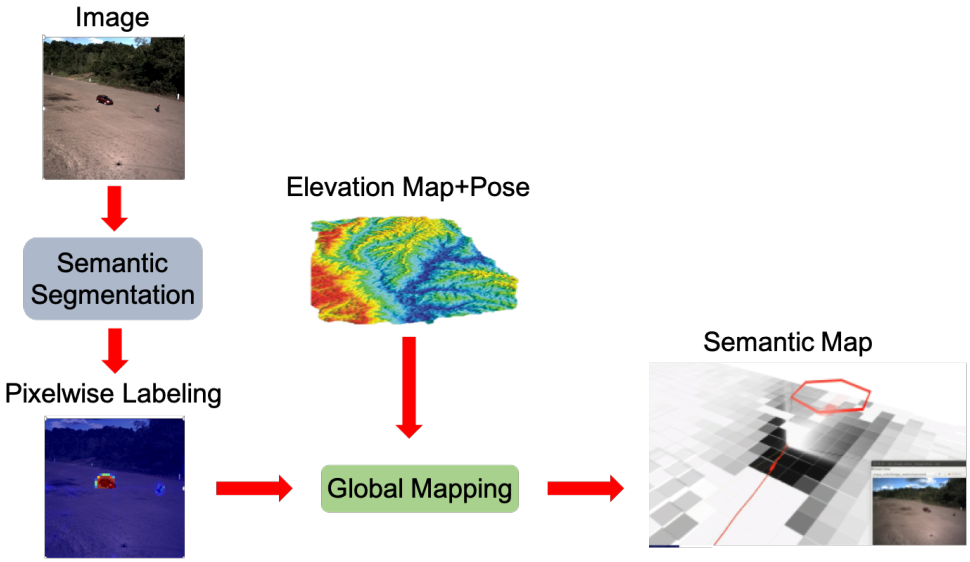


Figure 6.2: System outline of semantic mapping for scouting

To this end, the semantic mapping system must answer two questions about the scene: *what* objects of interest are in it, if any, and *where* are they, in physical space. To answer these questions, our semantic mapping module, depicted has in Figure 6.2, has two main stages. In the first stage, *semantic segmentation*, we use a deep learning system to label monocular camera imagery. In the second stage, *mapping*, we project the segmentation into a 2.5D grid map which maintains the robot's belief about the semantic class of each grid cell. We describe each stage in further detail in the following sections.

6.3.1 Lightweight semantic segmentation for aerial data

In the *semantic segmentation* stage, the goal is to assign one of K predefined semantic labels to each pixel in an RGB image. In this paper, the semantic classes are *car* and *background*, where the background class simply corresponds to anything that is not of interest. The choice of semantic classes was driven mainly by pragmatic reasons concerning our testing sites and available data, but the framework extends naturally to arbitrary semantic classes.

Semantic segmentation is closely related to *object detection*, for which the most common goal is to predict a bounding box around each instance of an object class. In this work, we prefer the pixel-level semantic segmentation approach over the detection approach, for several reasons: a) Current algorithms for segmentation are faster, b) We are interested in classes that may not be easily enclosed in a box, c) We do not require instance-level segmentation, d) It is trivially extended to multiple classes. Nonetheless, proposal-based approaches such as Faster RCNN [148], may present advantages for detection of small objects; this may be an interesting direction for future work.

However, as summarized in section 6.2, in recent years, the state of the art has been significantly advanced by Deep Learning, and in particular Fully Convolutional methods [113] which constitute our starting point.

To apply these networks in our project, we faced two challenges. First, we found the architectures to be too slow for real-time operation on our embedded platform. Second, we found that off-the-shelf architectures and datasets were optimized for ground-level, prominent objects in the image, whereas we are interested in distant objects that only occupy a few pixels.

Thus, for this project we created a custom architecture and dataset, as we describe below.

Architecture

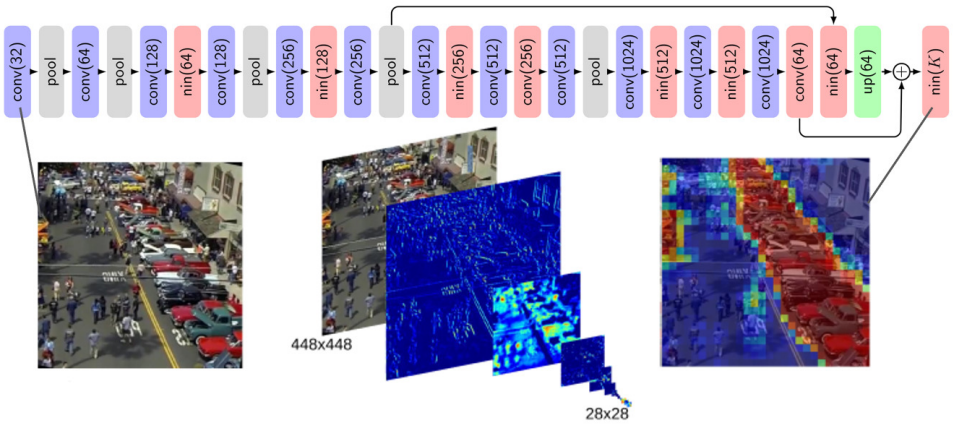


Figure 6.3: The ScoutNet architecture with example input and output

Our main architecture, ScoutNet, is shown in Figure 6.3. The structure is similar to FCNs. FCNs consist of a Directed Acyclic Graph (DAG) of convolutional and pooling layers, with a 3-channel RGB image as the input and a K-channel "label image" as the output, not necessarily the same size as the input. Given densely labelled images, a differentiable loss function such as cross-entropy can be computed for each output pixel, and since the whole network is differentiable, it can be trained via gradient descent using backpropagation. At runtime, inference for pixelwise labelling is purely feed-forward and can be performed efficiently with GPUs.

However, most work in this area has focused on maximizing accuracy, at the expense of memory and computing requirements. This becomes evident when applying these methods on relatively low-power platforms such as the Nvidia TK1, in which the inference times for the FCN-VGG16 from [113] proved to be in more than a second per image. Therefore, we made various experiments and modifications towards a faster architecture, even at the expense of accuracy.

Compared to FCN-VGG16, the first major difference in ScoutNet is the reduction of the number of filters in the initial layers, which was inspired by Tiny-YOLO [145], an object detection network. Another major change is the removal of input padding, which we found to cause a large increase in computational cost; in theory, it is useful to align receptive fields, but in practice [188] reports that removing has little effect. As in Tiny-YOLO, we also have a fully-connected layer that has global information. Unlike Tiny-YOLO, and like FCNs, we use skip layers. Finally, we also output a lower-resolution labeling. In the FCN, regardless of the effective classification resolution, the output is scaled to the original resolution at the end, even when training. Instead, we simply output the low-resolution output (1/16 of the input, in our case); during training and validation,

we downscale the label image. It must be noted that this slightly changes the objective, *e.g.*, it is possible for small labelled objects to disappear when downscaled.

Our choice was driven by the observation that increasing the output resolution through upsampling added significant cost to runtime inference by around 2×2 when doubling the resolution. We found transposed convolutions [113] to be computationally similar to unpooling plus convolution [132, 5]. However, for our purposes, a high output resolution is not essential, as long as we detect the object presence and approximate location. On the other hand, high *input* resolution is important, since smaller objects (in image space) are harder to detect; for highly downsampled images, many of the smaller objects simply disappear.

Here we adopt an 448×448 input resolution, in place of the 224×224 resolution commonly used by other architectures since AlexNet [98]. For the output resolution, we use 28×28, *i.e.*, a 16-pixel stride. To further use the high resolution provided by the 2MP camera onboard our vehicle, we extract a 896×896 center crop from each image and analyze it as four 448×448 patches, resulting in a 56×56 output. The patchwise classification is necessary due to memory constraints on the vehicle.

Dataset

To address the data issue, we created our own dataset. To reliably detect the classes of interest, we need to learn how they appear from the highly varied viewpoints and ranges we encounter in MAV data; but to our knowledge, there is no dataset for object detection or semantic segmentation for oblique, low-altitude ($10\,\mathrm{m}$ to $40\,\mathrm{m}$) aerial imagery. Instead, existing datasets feature top-down views (*e.g.*, VEDAI [143]) or are heavily biased towards ground-level imagery (*e.g.*, MS-COCO [111, 55]).

Fortunately, thanks to the recent popularity of camera-equipped consumer MAVs, thousands of aerial videos from around the world have been made publicly available on video-sharing websites such as YouTube and Vimeo. These videos vary widely in location, season, time of day, camera intrinsics, video quality, and so on, making for a diverse but challenging source of data.

We downloaded an initial dataset of approximately 1200 videos by searching video-sharing websites for various terms related to MAVs, including *drone*, *fpv* (short for First Person View), *aerial*, and names of popular consumer MAVs brands and models. As an initial exploration of this data, we extracted FC7 features for regularly sampled frames of each video using a VGG network [161] pretrained on the Places dataset [190], and clustered the feature vectors using k-means. We found the resulting clusters effectively grouped video segments according to the visually similar scenery type (*e.g.*, beaches, cities, parks, suburban areas) as well as clusters of videos with irrelevant subject matter (*e.g.*, reviews of MAV equipment, MAV-related news coverage, "drone" music concerts). Example clusters are visualized in Figure 6.4.

We then manually chose a diverse set of videos and manually labeled the cars in the dataset with polygons. We labeled 825 images from this dataset; while this is relatively small compared to recent datasets such as MS-COCO [111], it is comparable to datasets such as CamVid [16], which has been used to train networks from scratch [5]. We should also note the fairly high resolution of our images: 1920×1080, compared to less than 500 pixels per side for most other datasets. We call this the MAVCAR dataset.

Finally, we also create another dataset consisting of 500 images captured from our own field

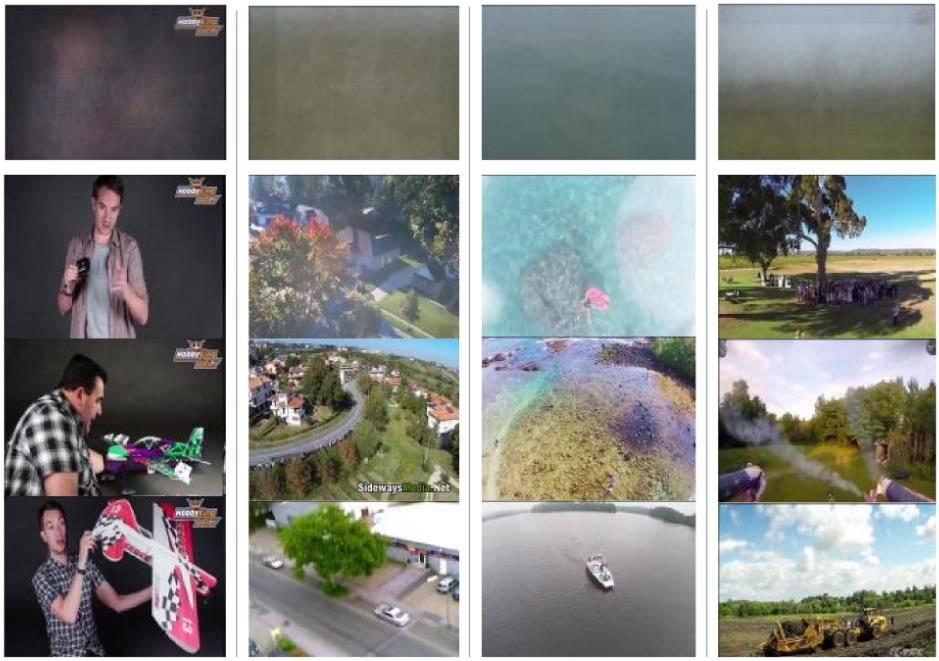


Figure 6.4: Frames belonging to videos in automatically extracted clusters, with one cluster in each column, and the top row showing the average cluster frame. The cluster on the left corresponds to videos of reviews for MAV hobbyists, which we discard.

experiments, spanning two years and two locations. Like before, we label cars only. We call this the FIELD dataset.

Example images from the datasets are shown in Figure 6.5.

Figure 6.5: Example images from our datasets

6.3.2 Mapping with digital elevation maps and prior knowledge

Given a semantically classified image, we want to find the position of objects detected in the image, as well as model regions for which the information in measurements is uncertain. Since this mapping has to be performed on board the vehicle, the driving requirement of the application is latency. Given a global pose by a state estimation filter, each pixel in the labeled image defines a ray originating at the camera center and passing through the pixel center. To perform the mapping operation, we use the images with soft pixel-wise predictions, together with the robot's global pose estimate and a pre-existing digital elevation map (DEM). We exploit the semantic knowledge of the world (every object rests on the ground) and use the digital elevation map to infer the 3D structure of the environment.

The efficiency of occupancy grid-based mapping algorithms has made them a popular choice for on-board processing in robots. The key to their efficiency is two simplifying assumptions: that the grid cell states are independent binary variables and that the measurements themselves are independent from each other, given the cells' true occupancy values. These assumptions have been shown to work effectively with range-bearing sensors.

However, a semantically classified image provides bearing-only measurements through rays originating from the camera pose, making the ray independence assumption limiting. To fully exploit the bearing-only measurement and the semantic structure knowledge of the world, we need to model ray interdependence. section 6.3.2 and section 6.3.2 describe how we model dependence amongst observations while still allowing for an on-line mapping algorithm.

Exploiting prior semantic knowledge

We assume that the objects of interest, represented by $\mathcal{L}_{\mathcal{M}} = \{c_1, c_2, ...c_n\}$, rest on the ground, and that we know their likely height $h_{c_i} \forall c_i \in \mathcal{L}_{\mathcal{M}}$. We model the world as a $2.5D$ grid. In every

cell C_{ij} of the grid at location i, j, we store the heights at which rays pass over the cell for all classes, by casting rays originating from the camera center of the classified image Table 6.1.

Table 6.1: Data members of grid cell C_{ij} for class c.

Symbol	Description
$C_{ij}(c, h_u)$	The highest height at which a ray with label c passes over or intersects the cell C_{ij}.
$C_{ij}(c, h_l)$	The lowest height at which a ray with label c passes over or intersects the cell C_{ij}.
$C_{ij}(c, n_f)$	The number of rays with label c that pass over or intersect the cell C_{ij} at a height less than h_c.
$C_{ij}(c, n_a)$	The number of rays with label other than c pass over or intersect the cell C_{ij} at a height less than h_c.
$C_{ij}(c, p_f)$	The cumulative probability of rays with label c that pass over or intersect the cell C_{ij} at a height less than h_c.
$C_{ij}(c, p_a)$	The cumulative probability of rays with label other than c that pass over or intersect the cell C_{ij} at a height less than h_c.
$C_{ij}(c, l_o)$	Integrated log-odds of an object of class c being present in the cell C_{ij}.

We are interested in finding cells where the height of rays passing over the cell matches the height of the object we are looking for, while accounting for occlusions and limited field of view. This leads to following cases for a given class in a cell C_{ij} and a class of concern c:

Case 0: The average probability of rays that pass over cell C_{ij} with a label other than class c is greater than the average probability of rays with class c.

Case 1: The rays of some other class pass from below and above the class c over the cell C_{ij}.

Case 2: The rays of some other class pass from below and nothing is observed above the class c over the cell C_{ij}.

Case 3: Nothing is observed above or below the class c over the cell C_{ij}.

Case 4: Nothing is observed below and some other class is observed above the class c over the cell C_{ij}.

Case 1 implies that the cell is well observed. Therefore, $C_{ij}(c, h_u)$ and $C_{ij}(c, h_l)$ should be close to h_c and the ground height respectively. **Case 2** implies that the upper part of the object could not be sensed due sensing geometry or occlusion. Hence, $C_{ij}(c, h_u)$ should be less than h_c and $C_{ij}(c, h_l)$ should be close to the ground. Similarly, **Case 3** implies that $C_{ij}(c, h_u)$ and $C_{ij}(c, h_l)$ should be less than h_c, whereas **Case 4** implies that $C_{ij}(c, h_u)$ should be close to h_c and $C_{ij}(c, h_l)$ should be less than h_c. These cases lead to Equation 6.1, that is used to determine whether there is positive, negative, or a lack of evidence in the current classified frame regarding the presence of an object of class c over the cell C_{ij}:

$$\phi_{ij}(c) = \begin{cases} 0, & \text{Case } 0 \\ \exp\left(\alpha_k C_{ij}(c, h_l)\right) \cdot \exp\left(\beta_k \dfrac{(C_{ij}(c, h_u) - h_c)}{h_c}\right) \cdot \dfrac{C_{ij}(c, p_f)}{C_{ij}(c, n_f)}, & \text{Case } k \end{cases} \tag{6.1}$$

Where $k \in [1, 4]$ and α_k, β_k are negative constants that allow us to change the weights of the measurements according to the cases encountered. We use the following hand-picked values for these constants: $\alpha_1 = \beta_1 = -10, \alpha_2 = -10, \beta_2 = -1, \alpha_3 = -1, \beta_3 = -1, \alpha_4 = -1, \beta_4 = -10$. The resulting values $\phi_{ij}(c)$ encode evidence for the presence of class c in the cell (i, j). The values are in the range $[0, 1]$ and values that are larger, smaller or close to 0.5 indicate positive, negative, and lack of evidence respectively.

Temporal Evidence Integration

The $\phi_{ij}(c)$ term allows the algorithm to model the dependence amongst rays, while allowing us to treat the cells independently. We assume that for any given cell, the log-odds of the probability of observing a class c is given by a constant γ. Each class in a cell is represented as an independent binary random variable, as a cell can have objects of multiple classes. Once the value of evidence $(\phi_{ij}(c))$ is identified, the log-odds for each class in each cell are updated with equation Equation 6.2.

$$C_{ij}(c, l_o) = \begin{cases} C_{ij}(c, l_o), & |\phi_{ij}(c) - 0.5| \leq \zeta \\ C_{ij}(c, l_o) + \gamma \left(C_{ij}(c, n_f) - C_{ij}(c, n_a) \right), & \text{otherwise} \end{cases} \tag{6.2}$$

Where ζ is a small positive number less than 0.5. We use $\zeta = 0.2$ and $\gamma = 1$. As the MAV performs its mission and captures images, each image is semantically segmented with our network and associated with the estimated camera pose at the time of capture. As soon as each image is segmented, the labels in the segmentation and the estimated pose are used to update the grid and $C_{ij}(c, l_o)$.

6.3.3 System implementation

Figure 6.6: Our MAV platform

Our current MAV is depicted in Figure 6.6. The base platform is an off-the-shelf quadrotor DJI vehicle retrofitted with our own sensors and computing payload designed for autonomous scouting.

The system architecture is summarized in Figure 6.7.

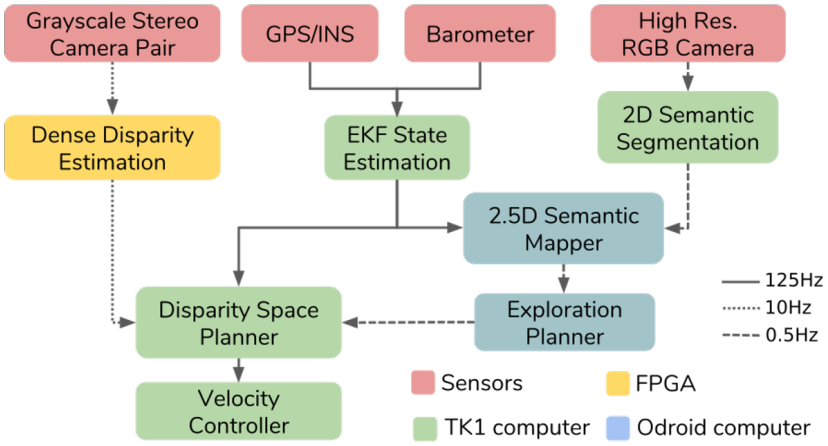


Figure 6.7: Block diagram of the MAV scouting system

Sensing

The sensor suite consists of a monochrome stereo camera pair, a monocular color camera, an integrated GPS-INS unit and a barometer. The GPS/INS system and the barometer are used for state estimation.

All cameras are forward-facing, tilted downwards at $15°$. an orientation well suited for low-altitude ($<40\,\mathrm{m}$) operation. The horizontal field of view for this camera is approximately $60°$, which we considered a good compromise between coverage and object size, given the sensor resolution of 1600×1200 pixels.

Hardware platform

All computation for autonomous operation is performed on board, using two embedded ARM computers. The first is an NVidia TK1, which features a low-powered GPU which we use for semantic segmentation. This computer also runs other perception-related tasks necessary for autonomous operation. The second is an ODroid XU4, which runs the mapping and planning systems.

Software platform

Both computers use ROS on Ubuntu Linux. Our segmentation and mapping methods run concurrently as ROS nodes and communicate through messages. The segmentation node, implemented in Python, uses the Theano [8] library with the Nvidia CuDNN backend to make effective use of the GPU. The mapping algorithm is CPU-only and is implemented in C++.

6.4 Results

Here, we present results for each of the two main subsystems in isolation, and document the integrated system performing a fully autonomous mission in the field. Finally, we also discuss

extensions to the current system that address some of its current limitations.

6.4.1 Semantic segmentation evaluation

We study the performance of ScoutNet, two ablations, and 16-stride FCN-VGG16 on the Pascal-Context dataset [124], a popular semantic segmentation dataset with around $10\,000$ densely labelled images. The first ablation, ScoutNet-nopool, replaces all pooling layers with convolutional striding, for a slight speedup. The second, ScoutNet-noskip, removes skip layers, resulting in a pure encoder-decoder architecture in the style of SegNet [5]. For FCN we evaluate at full resolution, the standard, and at the reduced resolution, like our models. We implemented all networks, but FCN-VGG16 was ported from the author's implementation to Theano [8], including the weights. The FCN methods are included as a baseline, since in practice they are too slow for our application.

The results are shown in Table 6.2. As a sanity check, standard FCN-VGG16 is within 2% of the reported results in [113]. This table also shows a drop in performance when reducing the output resolution (`fcn-vgg16-lr`), an expected accuracy-performance trade-off. Finally, we see the impact of ScoutNet ablations, where removing pooling or skip layers improves speed but hurts performance.

Table 6.2: Performance on Pascal-Context Validation

Method	Pixel Acc.
`fcn-vgg16`	**0.65**
`fcn-vgg16-lr`	0.57
`scoutnet`	0.45
`scoutnet-nopool`	0.39
`scoutnet-noskip`	0.33

We then evaluate on our MAVCAR dataset, consisting of 410 frames from 155 different videos. We use 328 training frames from 101 videos, and the rest of the data is used for validation. We ensure that there are no videos shared between the training and testing sets to avoid overfitting. All images are 1920×1080, so we follow an atypical protocol for training and validation. At training time, we sample a random crop of size 448×448 from anywhere in the image. At test time, for each image we create 896×896 center crop, divide it in four 448×448 patches, and evaluate each separately. This mimics the inference processed used in our live experiments. Figure 6.8 shows examples. We can see, again in Table 6.3a that ScoutNet achieves worse performance than FCN. On the other hand, results are qualitatively acceptable Figure 6.8. Finally, it evaluate the importance of using aerial versus ground-level data, we trained ScoutNet on a 2588-image subset of MS-COCO [111] with vehicles, and tested on FIELD. We repeated the process with MAVCAR. As Table 6.3b shows, MS-COCO, despite having more images, is less useful for our task due to its ground-level bias.

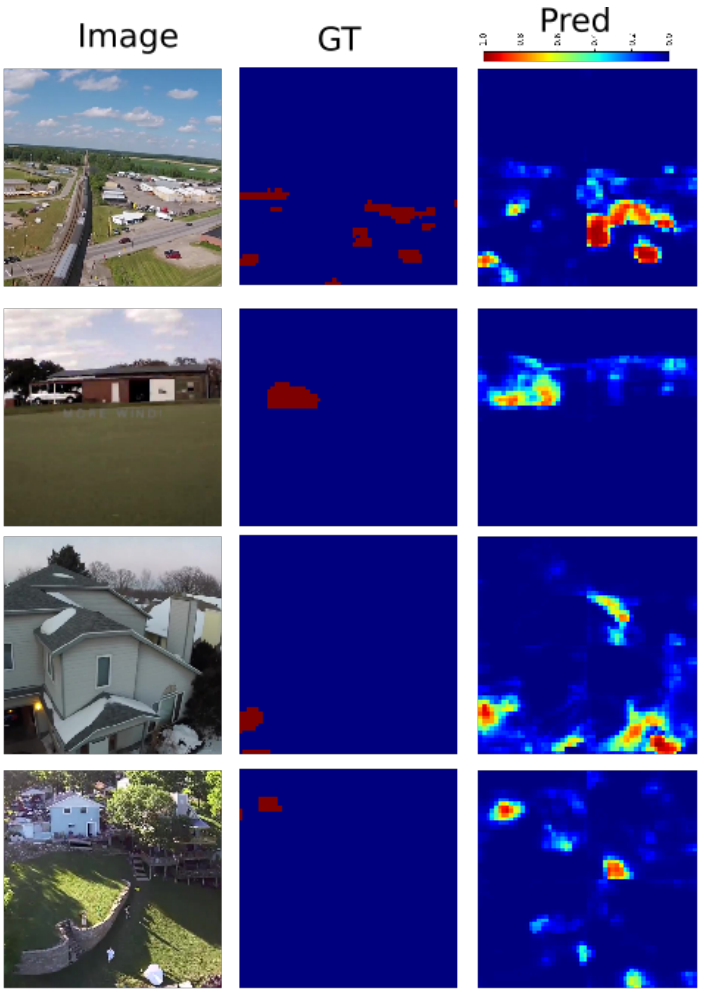


Figure 6.8: Car predictions from ScoutNet on the MAVCAR test set, as heat maps. Middle column is ground truth. The two bottom rows show failures.

Table 6.3: Quantitative evaluation on the MAVCAR and FIELD datasets

(a) Performance of different architectures on MAVCAR validation dataset

Variant	Prec.	Recall	IoU
`scoutnet`	0.43	0.49	0.30
`vggs-fcn`	**0.53**	**0.58**	**0.38**

(b) Effect of Training Dataset when testing on FIELD Dataset, for ScoutNet

Training Set	Prec.	Recall	IoU
MS-COCO Vehicles	0.12	**0.75**	0.11
MAVCAR Train	**0.53**	0.30	**0.24**

Qualitative field results

Figure 6.9 shows results of the system operating on human-piloted field data, along with a top-down map depicting the estimated position of cars obtained with the system described in subsection 6.3.2.

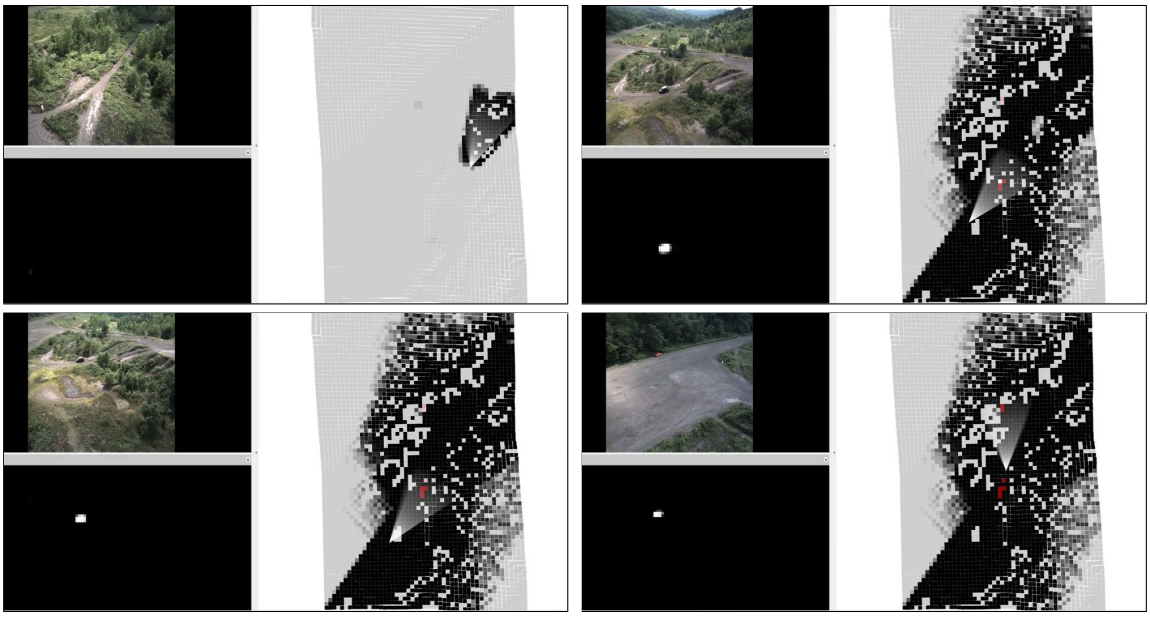


Figure 6.9: Screen captures of field results on a human-piloted flight. In each frame, the left shows the input image (top left) and the output of ScoutNet (bottom right; white pixels indicate car detections). The right side shows a top-down map of estimated car positions (as red squares) using the method described in subsection 6.3.2.

Timing

Segmentation is currently the bottleneck in our pipeline. We evaluated the average per-frame time to classify four 448×448 patches in each network. The classification must be performed four times to evaluate the desired 896×896 region. We use the embedded TK1 platform described in subsection 6.3.3.

With this setup, we find that the FCN-VGG16 takes more than $600\,\mathrm{ms}$ per patch, or more than $2.4\,\mathrm{s}$ per image; this renders it unusable for our purposes. In contrast, ScoutNet takes around $280\,\mathrm{ms}$ per patch, and around $750\,\mathrm{ms}$ per image, as we evaluate the four patches within a single batch. Due to memory limitations we found this impossible with the FCN-VGG16. The large difference in performance leads us to use ScoutNet over FCNs, despite its inferior accuracy.

When adding time for preprocessing (100 ms) and slower operation due to the load caused by other processes during flight, in practice we find the effective labelling rate to $0.5\,\mathrm{Hz}$. This is slower than we would like, but usable in low-speed operation.

6.4.2 Mapping

In this section, we demonstrate the effects of exploiting semantic knowledge and modeling ray dependence qualitatively, while measuring the sensitivity of the mapping algorithm to height inaccuracies in the DEM.

Figure 6.10-4 shows a canonical scenario where a car, more than $50\,\mathrm{m}$ away, is detected by the semantic classification algorithm. Exploiting semantic knowledge and modelling dependence allows the mapping algorithm to capture the uncertainty about the presence of a car in the cell occluded by the car (Figure 6.10-1), whereas if we do not reason about ray interdependence, the occluded cell is also inferred to contain cars (Figure 6.10-3). If both the semantic knowledge and ray interdependence are not exploited, then a simple projection of classified image to DEM leads to an inference that multiple cells are occupied by a car (Figure 6.10-2). This case shows how modelling the ray interdependence and exploiting semantic knowledge leads to better mapping of objects and uncertainties. On the other hand, Figure 6.10 shows that the algorithm's performance deteriorates in the presence of height errors in the DEM. We can see that the degradation is faster if the DEM underestimates the height of the cells, due to the geometry of the observations.

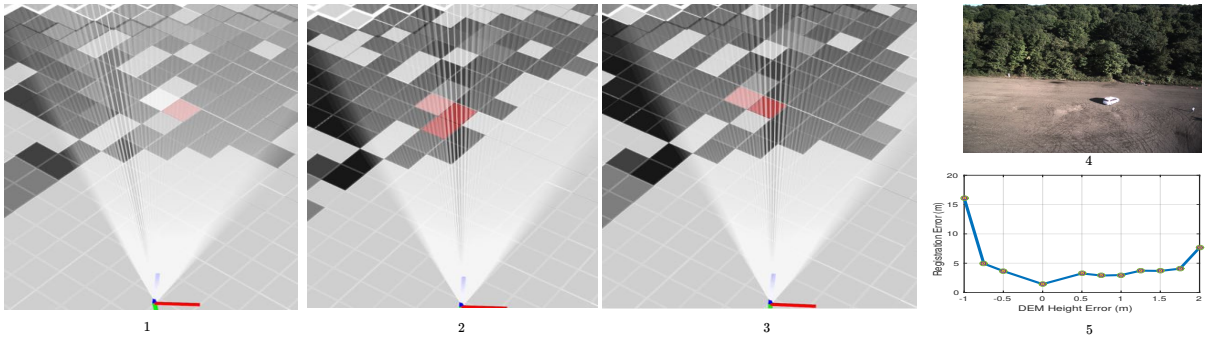


Figure 6.10: Analysis of our mapping method in a representative scenario. Figure 4 shows the image observation integrated in our current mapping pipeline. Figure 1 shows the updated map after a classified image. Figure 2 shows the updated map if the classified image is projected on the DEM without exploiting semantic knowledge, and figure 3 shows the updated map if the ray interdependence is not modelled. Dark gray squares indicate absence of cars and red squares the presence of cars; intermediate shades of grey and red reflect uncertainty. Modelling ray interdependence and exploiting semantic knowledge leads to better modelling of uncertainties due to occlusions, while providing an improved cell occupancy estimate. Figure 5 provides the sensitivity analysis of mapping performance versus DEM height errors.

6.4.3 Field Results

In this section, we show scouting mission conducted by the MAV platform using the classification mapping pipeline described in this chapter. Figure 6.11 and Figure 6.12 show two autonomous missions, in each of which the vehicle is deployed to scout for cars and collect high resolution data if a car is found. In both missions, the semantic segmentation and mapping algorithm is able

to detect and map both cars in the environment with sufficient accuracy to enable collection of high resolution data of the said cars.

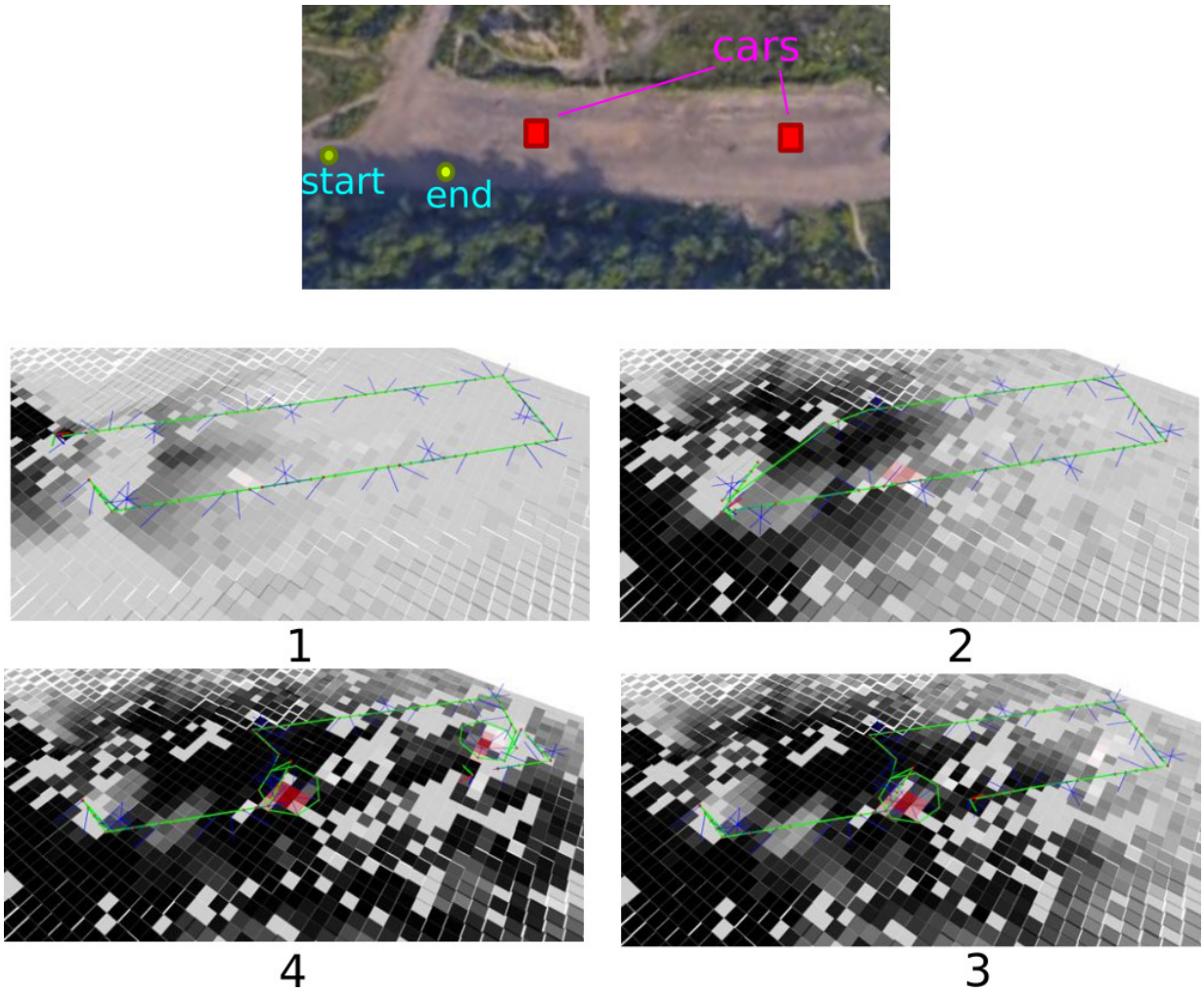


Figure 6.11: Autonomous scouting mission. On the top, testing site, start and end are marked by green nodes and car locations are shown in red. Figures 1–4 show the MAV's plans at various stages of the exploration mission. Dark gray squares indicate absence of cars and red squares presence of cars, with intermediate shades reflecting uncertainty. Once the car is recognized, a 360° view of the car is obtained. The mapping pipeline enables detection and data collection for both cars present in the environment.

6.4.4 Extensions

Multi-class predictions

In the results we have shown so far, we label a single class, *car*, mainly due to the cost of labeling data for more classes. However, our pipeline supports multiple classes. In Figure 6.13, we show the

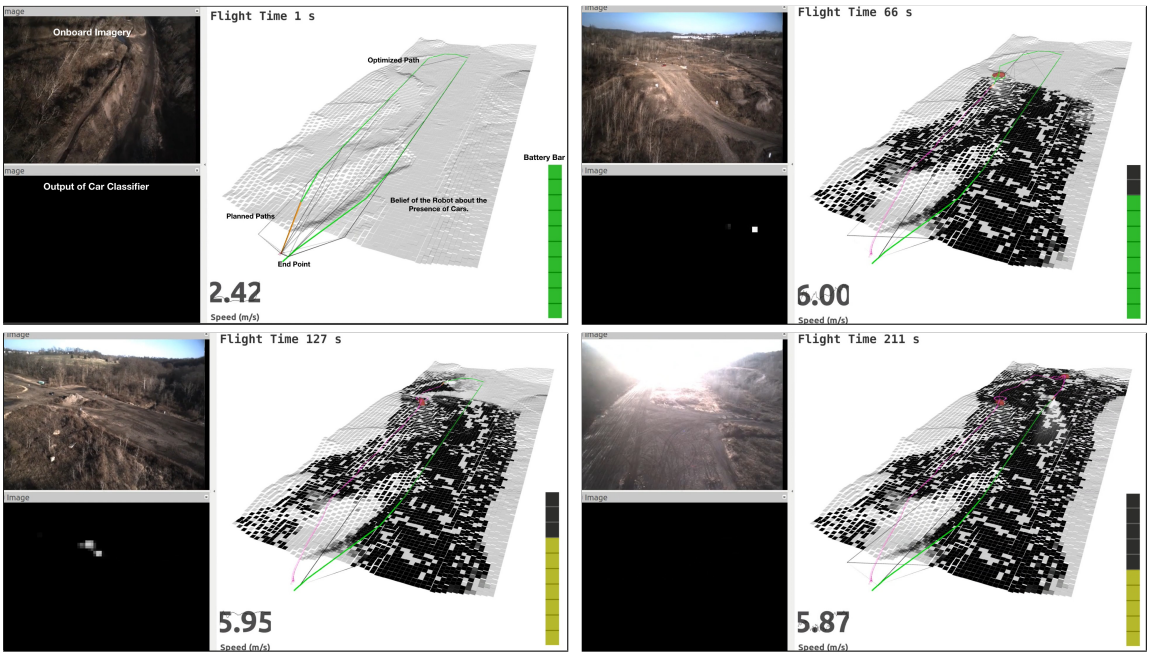


Figure 6.12: Larger scale autonomous scouting mission. In each frame, the left shows the input image (top left) and the output of ScoutNet (bottom right; white pixels indicate car detections). The right side shows an oblique view of the estimated car positions (as red squares), along with the planned and actual trajectories of the vehicle during the mission.

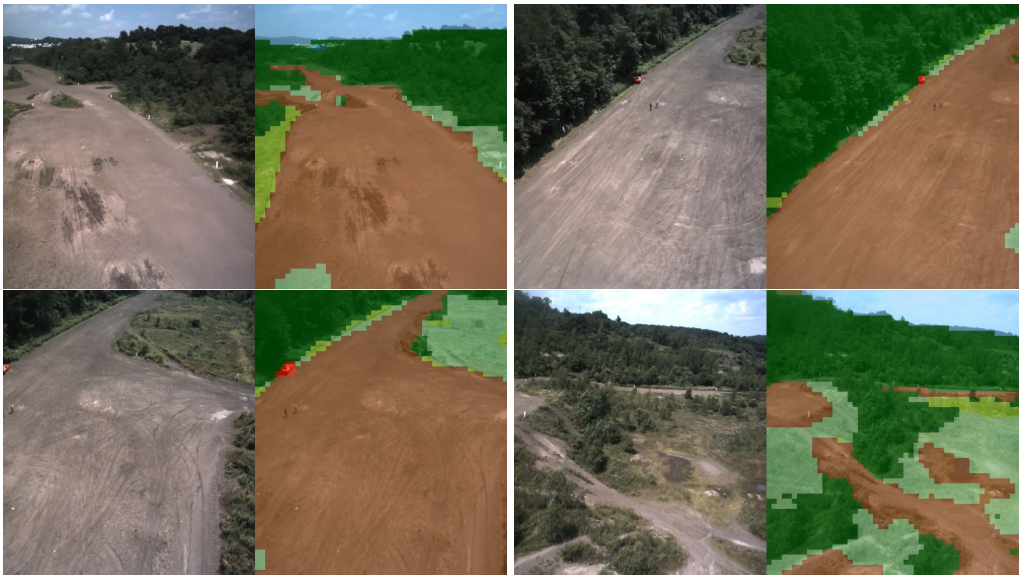
results of training our semantic segmentation with a combined dataset including Pascal-Context and our own custom datasets.

DEM-free operation

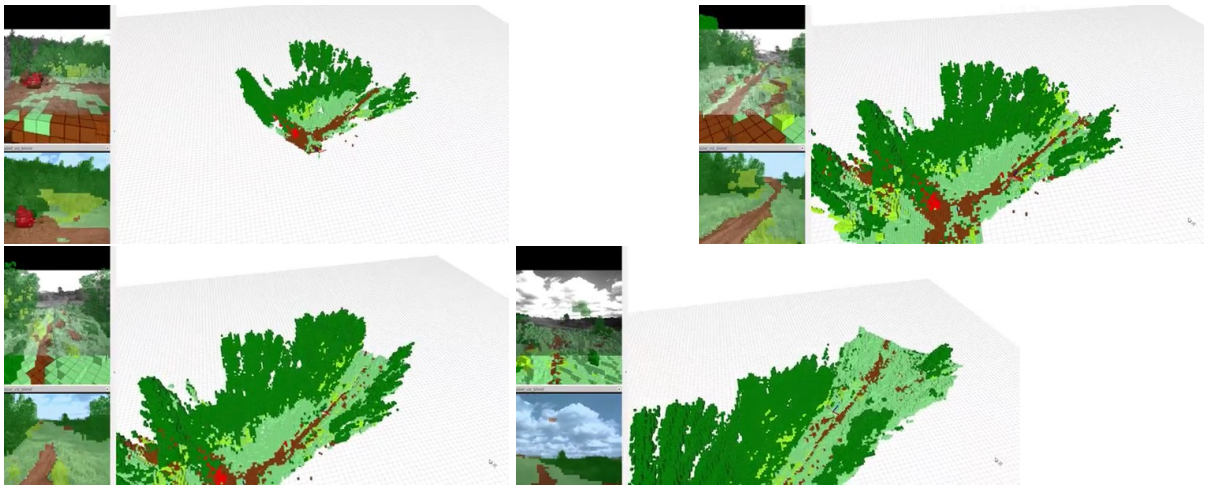
However, as we explored multiclass segmentation, especially with higher spatial resolution maps than in our earlier work, we observed multiple issues caused by drift in the state estimation, noisy depth estimates from stereo and inaccuracies in the DEM. To mitigate these issues, we explored a learning-based approach to correct for inaccurate stereo depth by creating a training dataset using Structure-from-Motion (SfM), metrically scaled with GPS information (Figure 6.14). In this approach, we no longer use DEMs, but instead use only our learned depth predictions. Moreover, we use pose estimates derived from visual SLAM in combination with the GPS-INS to reduce drift, and fuse depth measurements into the elevation map in a robust fashion with per-cell running median filters. Figure 6.15 shows preliminary results of this approach.

6.5 Summary

In this chapter, we have described a semantic mapping system aimed to support autonomous scouting with MAVs. We evaluated the two main components of the system in isolation and demonstrated semantic exploration in integrated autonomous missions.



(a)



(b)

Figure 6.13: (a): Multiclass semantic labeling (2D only). (b): Multiclass semantic labeling (2D projected on 3D)

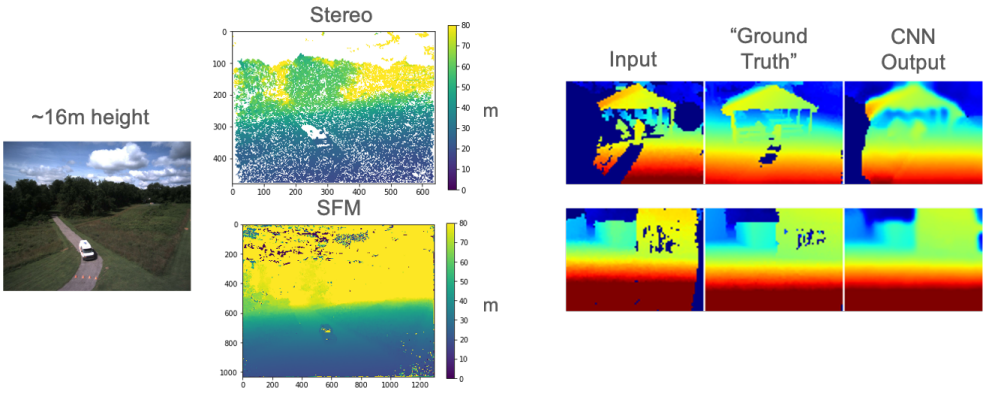


Figure 6.14: Learning to correct noisy and sparse stereo disparity using SfM reconstructions as a source of supervision. In the left, an RGB image captured from the MAV is shown, along with a visualization of the corresponding pixelwise depth as measured by the onboard stereo camera and by an off-line SfM reconstruction, scaled with GPS-INS metadata. The SfM cloud is denser and more accurate at long ranges. In the right, we show examples outputs of a CNN trained to interpolate and refine the noisy disparity images using the SfM point clouds as an approximation to "ground truth".

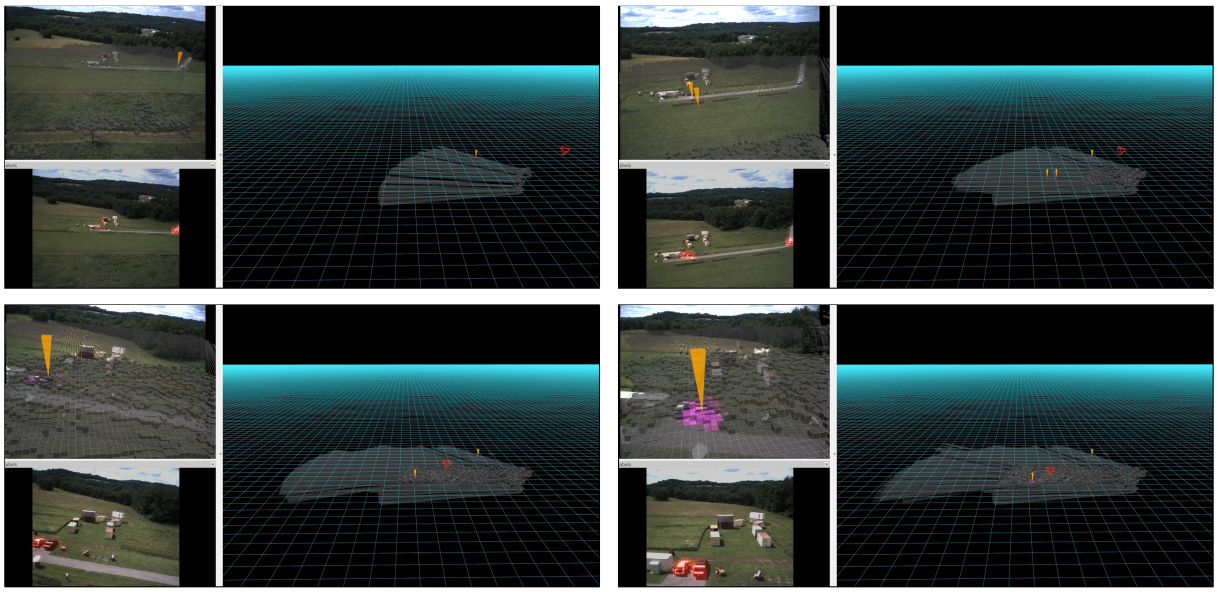


Figure 6.15: Detecting vehicles at long distance without DEMs by using VSLAM and learned depth refinement

However, there are several limitations in the current system. Its performance is often limited by inaccuracies in the depth estimation, the estimated 3D pose and the DEM, which become particularly visible over long ranges. While subsection 6.4.4 explores some preliminary directions to mitigate these issues, it is still an unsolved problem. We believe that strong reliance on GPS-INS and DEMs is a suboptimal solution, especially for vehicles equipped with lower-grade sensors. Similarly, naive stereo sensing is also limited for this problem, given its quadratic degradation of accuracy with distance. Going forward, solving this problem may require a deeper integration of visual SLAM in the semantic mapping pipeline, as well as the incorporation of deep learning advances in monocular and stereo depth estimation (*e.g.*, [91, 114]) and a robotic platform with the additional computational power to handle these tasks.

Since the publication of our work in this area, there have been some advances in SLAM that can potentially address some of the issues we have encountered in terms of state estimation. An example is the work of Cao, Lu, and Shen [22], who propose a principled non-linear optimization approach to state estimation from GNSS, visual odometry and inertial sensing.

A relevant research area that has become highly active after the publication of our work is semantic exploration. A large portion of this work has been spurred by benchmarks for this task using realistic simulated indoor environments, where the goal is for a simulated robot agent to search for an object of a certain class; a recent example is Chaplot et al. [27]. While this indoors, simulated task presents different challenges than the (non-simulated) outdoors task with an MAV, they offer interesting ideas for learning-based approaches to semantic exploration that would be interesting to explore for our scenario.

Chapter 7

Conclusions and future directions

In this thesis, we have shown how semantic maps can be used as a richer inner representation to extend and improve the capabilities of mobile robots operating in the real world. In the process, we make several contributions to the state of the art in semantic mapping with image and point cloud data:

- We develop a novel approach for semantic mapping of low-altitude aerial point clouds, providing an autonomous helicopter with awareness of various semantic classes relevant to navigation and landing in unprepared areas. The approach features a novel ground surface estimation step which is robust to the occlusion patterns of aerial point clouds, and is used to create discriminative point cloud features for semantic segmentation with a learned classifier. The novel ground surface feature is shown to improve classification accuracy in a dataset captured from a helicopter. Additionally, the ground surface estimate is used to estimate the landing zone height from noisy measurements at long distances (chapter 3).
- We propose a novel deep learning approach for 3D data that obtains superior classification accuracy relative to prior approaches based on hand-engineered features. Our proposed approach learns 3D CNNs that jointly perform feature extraction and classification from a volumetric representation. We show improvements over the state of the art in landing zone detection and object recognition (chapter 4).
- We propose two systems for multimodal semantic classification from 2D image data and 3D point cloud data. The first system proposes a novel 2D-3D CNN architecture for the joint use of these two modalities in learned semantic inference tasks. The architecture outperforms various unimodal and multimodal baselines in a semantic segmentation benchmark with lidar and image data. Our second system proposes a more efficient decoupled usage of these two modalities for semantic mapping, using image data for semantic inference and lidar data for spatial inference. This system achieves real time operation and is tested in the field, allowing an autonomous All-Terrain Vehicle to traverse grass and narrow trails that would be considered untraversable by a system using purely geometric maps. (chapter 5).
- We propose a long-range, image-centric semantic mapping system that does not use range sensing for semantic inference. The system can build coarse semantic maps of distant objects using monocular images and INS positioning data by leveraging publicly available digital elevation maps. We use this system for the task of semantic exploration with a Micro-Aerial

Vehicle, enabling significant time savings in the autonomous gathering of data for a specific semantic category (chapter 6).

In each case, we have applied semantic mapping to extend the capabilities of real robots operating in the field, showing the benefits of a richer semantic understanding of the world compared to purely spatial representations.

Despite these positive results, we have observed several limitations to our current framework, that are related to two significant choices we make in this thesis: using semantic categories and global metric maps.

As we argued earlier, these choices are beneficial for the purposes of allowing robots to build semantic maps that are interpretable for humans and are highly compatible with the dominant paradigms in planning and machine learning. However, throughout the development thesis we have also faced challenges that stem from these choices.

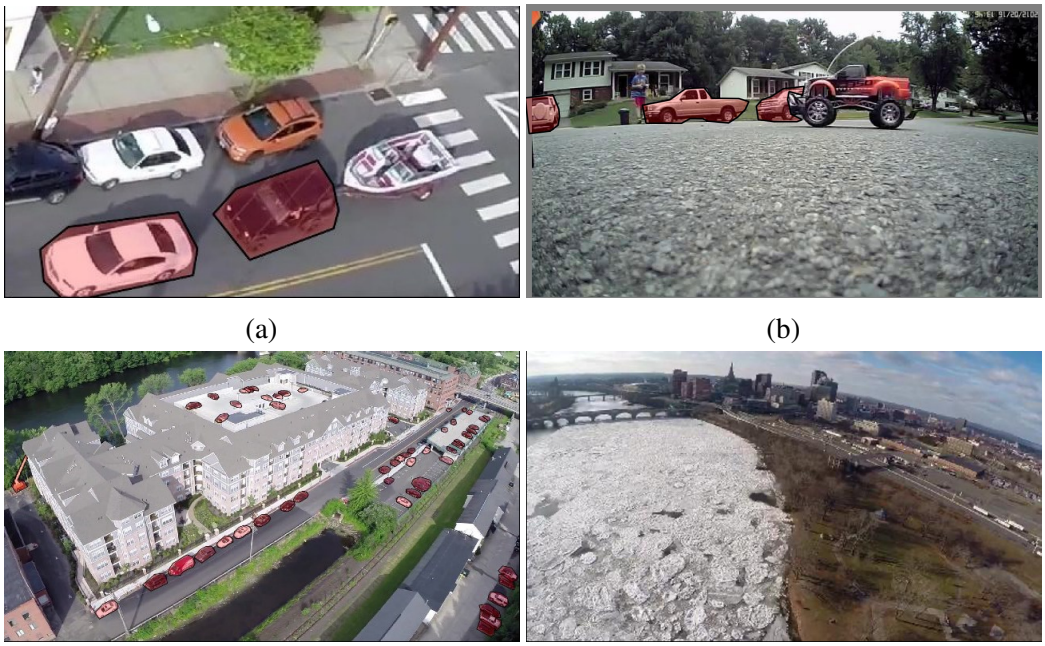
Issues with semantic categories As described in chapter 1, the systems developed in this thesis describe the semantics of the environment in terms of a finite set of categories, which are chosen in an *ad hoc* way for each problem.

However, the question of "how to carve nature at its joints" for any given robotic task is in general non-trivial. While we have been able to identify human-interpretable semantic categories that have proven useful for each of the applications featured in this thesis, in each case the process was far from painless. In practice, it is often difficult to specify semantic categories for a semantic mapping system, as it requires consideration of multiple constraints concerning different robotic subsystems, as well as other engineering factors. The chosen semantic categories must be possible to discern from the available data, for both the robot and humans (when manual labeling is required). The categories must also provide relevant information for the planning system that will use the semantic map to accomplish the system's goals. In many cases, it is necessary to co-design the sensing and planning systems in order to meet these conditions, resulting in considerable engineering effort. The problem is compounded if the structure of the tasks to be accomplished is not well-defined, as often happens in practice.

Semantic categories are often ill-defined or ambiguous. For example, one of the goals for the semantic mapping system for the ATV from chapter 5 was to allow the vehicle to not only follow a trail, but to avoid, when possible, "rough" trail sections in favor of "smooth" sections. In practice, specifying this category was difficult, not solely because "smooth" versus "rough" is a continuum rather than a binary distinction, but because human judgments of these properties proved to be only loosely correlated with objective criteria such as geometric roughness, depending also on perceived soil composition, presence of pebbles, slope, soil moisture, etc. Even categories that might appear relatively clear-cut, such as *car* or *ground* have abundant corner cases; see, *e.g.*, Figure 7.1 from chapter 6.

Furthermore, it should be kept in mind that the semantic map is just one part of a larger system. For any given choice of semantic categories, it is often challenging to design a planning system to take advantage of this map to accomplish the robot's task. For example, the implementation of the reactive planner of chapter 5 required various rounds of hand-tuning of the function to create cost maps from semantic maps.

Finally, it is also expensive and time-consuming to manually label data, and it is difficult to



(a) (b)

(c) (d)

Figure 7.1: Edge cases in manual labeling. (a) Is the trailing speedboat a vehicle? (b) Is the RC car a vehicle? (c) Is the rooftop parking lot a ground surface? (d) Is the frozen ice a ground aurface, or water?

predict how much data is necessary. In fact, given the dynamic nature of the real world, it is often necessary to make manual labeling a continuous task.

Issues with global metric maps Global metric maps also have issues. For example, we have observed that a purely metric representation of space is difficult to use in scenarios where semantic maps span large distances, require reasoning about multiple frames of reference with possibly conflicting information, and have large amounts of spatial uncertainty. We observed all of these conditions in chapter 6. In our deployed system, we make our best effort to use a global metric frame to represent absolute positioning estimates from GPS, relative pose estimates from visual odometry and IMUs, elevation estimations from DEMs, and the semantic segmentation and depth estimates from the on-board cameras. This leads to a metric representation that is readily usable by planning systems that require this representation, but leads to several difficulties in the face of uncertainty and noise. While it is possible to mitigate these issues in various ways, taking a step back, the way we have posed this task seems somewhat unnatural. If we see a car in the distance, do we need to know its GPS coordinates — or even its position relative to the robot in meters — to plan a trajectory flying towards it? Currently, our system does, but a human pilot would certainly not. While human perception of space is not completely understood, evidence (and subjective phenomenology) suggests humans dynamically switch between reference frames according to context [136, 99]. Currently, it is not obvious how to endow robots with this kind of flexibility.

Future directions

In the last few years, many lines of research have emerged that address some of the problems we have discussed. For example, approaches based on deep reinforcement learning have been proposed to enable an agent to learn how to act using raw sensor data as input, and with no other supervisory signal than a reward function [108]. In theory, this kind of approach allows the robot to learn its own representation of the world from scratch, without any engineering of localization and mapping systems or labeling of semantic categories. While this is an attractive idea, in practice it is difficult to apply for real world systems. As this approach requires the robot to learn by trial and error, for most robots the only way to learn through reinforcement learning is in simulation. However, it is often difficult to make the policies learned in simulation work in reality, as we learned first-hand in our own attempts to apply reinforcement learning for the self-driving task from chapter 5 [34]. In addition, it is still unclear how robots trained from scratch via reinforcement learning can be adapted to interface with humans; not only in terms of following human instructions, but also in terms of providing an interpretable view into their inner workings, so that the behavior of the robot can be predicted and ultimately, trusted to be reasonably safe.

Thus, while we believe the specific approach of using semantic categories and global metric maps is arguably too restrictive and brittle for many applications going forward, a fully end-to-end, "pixels-to-torque" approach to learning robotic behavior is also impractical. Given that biological agents do not learn everything from scratch, but rather, have been shaped through millenia by evolution, it seems reasonable for humans to invest a few years into engineering the right kind of representations into robotic agents. Ideally, these representations should provide robots with a certain amount of prior structure regarding the nature of the world, while still being flexible enough to allow the robots to efficiently learn and adapt from training data or interaction with the world. Interesting recent approaches along these lines include Cartillier et al. [25] and Casas, Sadat, and Urtasun [26]. Naturally, looking forward, there are many other possible approaches, and what these will look like is still an open question.

References

[1] John Amanatides and A Woo. “A fast voxel traversal algorithm for ray tracing”. In: *Proceedings of EUROGRAPHICS* i (1987).

[2] D. Anguelov, B. Taskar, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng. “Discriminative Learning of Markov Random Fields for Segmentation of 3D Scan Data”. In: *CVPR*. Vol. 2. IEEE, 2005, 169–176 vol. 2.

[3] David Anthony, Sebastian Elbaum, Aaron Lorenz, and Carrick Detweiler. “On crop height estimation with UAVs”. In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 4805–4812.

[4] Sankalp Arora and Sebastian Scherer. “PASP: Policy Based Approach for Sensor Planning”. In: *Proceedings of (ICRA) International Conference on Robotics and Automation*. IEEE, May 2015, pp. 3479–3486.

[5] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling”. In: (2015). arXiv: 1505.07293.

[6] Timothy D. Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017.

[7] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. “Algorithms for Hyper-Parameter Optimization”. In: *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*. Ed. by John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger. 2011, pp. 2546–2554.

[8] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. “Theano: a CPU and GPU Math Expression Compiler”. In: *SciPy*. 2010.

[9] Julian Besag. “Spatial Interaction and the Statistical Analysis of Lattice Systems”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 36.2 (1974), pp. 192–225.

[10] Andreas Bircher, Kostas Alexis, Michael Burri, Philipp Oettershagen, Sammy Omari, Thomas Mantel, and Roland Siegwart. “Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics”. In: *ICRA*. IEEE. 2015.

[11] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. USA: Oxford University Press, Inc., 1995.

[12] Michael Bloesch, Sammy Omari, Marco Hutter, and Roland Siegwart. “Robust visual inertial odometry using a direct EKF-based approach”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 298–304.

[13] Michael Bosse, Robert Zlot, and Paul Flick. “Zebedee: Design of a Spring-Mounted 3-D Range Sensor with Application to Mobile Mapping”. In: *IEEE Transactions on Robotics* 28.5 (2012), pp. 1104–1119.

[14] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (2001), pp. 5–32.

[15] André Brock, Theodore Lim, James M. Ritchie, and Nick Weston. “Generative and Discriminative Voxel Modeling with Convolutional Neural Networks”. In: *CoRR* abs/1608.04236 (2016). arXiv: 1608.04236.

[16] G Brostow, J Fauqueur, and R Cipolla. “Semantic object classes in video: A high-definition ground truth database”. In: *Pattern Recognition Letters* 30.2 (2009), pp. 88–97.

[17] Mitch Bryson, Alistair Reid, Fabio Ramos, and Salah Sukkarieh. “Airborne vision-based mapping and classification of large farmland environments”. In: *Journal of Field Robotics* 27.5 (2010), pp. 632–655.

[18] Wolfram Burgard and Martial Hebert. “World Modeling”. In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 853–869.

[19] Couprie C, C Farabet, L Najman, and Y LeCun. “Toward Real-time Indoor Semantic Segmentation Using Depth Information”. In: vol. 1. 2000, pp. 1–22. arXiv: arXiv:1301.3572v2.

[20] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. “Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age”. In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332.

[21] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J.J. Leonard. “Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age”. In: *IEEE Transactions on Robotics (TRO)* (2016).

[22] Shaozu Cao, Xiuyuan Lu, and Shaojie Shen. *GVINS: Tightly Coupled GNSS-Visual-Inertial Fusion for Smooth and Consistent State Estimation*. 2021. arXiv: 2103.07899 [cs.RO].

[23] M. Carlberg, P. Gao, G. Chen, and A. Zakhor. “Classifying urban landscape in aerial LiDAR using 3D shape analysis”. In: *2009 16th IEEE International Conference on Image Processing (ICIP)*. 2009, pp. 1701–1704.

[24] Matthew Carlberg, James Andrews, Peiran Gao, and Avideh Zakhor. *Fast Surface Reconstruction and Segmentation with Ground-Based and Airborne LIDAR Range Data*. Tech. rep. UCB/EECS-2009-5. EECS Department, University of California, Berkeley, 2009.

[25] Vincent Cartillier, Zhile Ren, Neha Jain, Stefan Lee, Irfan Essa, and Dhruv Batra. “Semantic mapnet: Building allocentric semanticmaps and representations from egocentric views”. In: *arXiv preprint arXiv:2010.01191* 2 (2020).

[26] Sergio Casas, Abbas Sadat, and Raquel Urtasun. “MP3: A Unified Model To Map, Perceive, Predict and Plan”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 14403–14412.

[27] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. “Learning To Explore Using Active Neural SLAM”. In: *International Conference on Learning Representations (ICLR)*. 2020.

[28] Ken Chatfield, Victor S. Lempitsky, Andrea Vedaldi, and Andrew Zisserman. “The devil is in the details: an evaluation of recent feature encoding methods”. In: *British Machine Vision Conference, BMVC 2011, Dundee, UK, August 29 - September 2, 2011. Proceedings*. Ed. by Jesse Hoey, Stephen J. McKenna, and Emanuele Trucco. BMVA Press, 2011, pp. 1–12.

[29] Nesrine Chehata, Li Guo, and Clément Mallet. “Airborne LIDAR feature selection for urban classification using random forests”. In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 38 (Jan. 2009).

[30] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. “Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[31] Tongtong Chen, Bin Dai, Daxue Liu, and Jinze Song. “Performance of global descriptors for velodyne-based urban object recognition”. In: *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*. IEEE, 2014, pp. 667–673.

[32] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. “Multi-view 3D object detection network for autonomous driving”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 1907–1915.

[33] Changhyun Choi, Wilko Schwarting, Joseph DelPreto, and Daniela Rus. “Learning Object Grasping for Soft Robot Hands”. In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2370–2377.

[34] Po-Wei Chou, Daniel Maturana, and Sebastian A. Scherer. “Improving Stochastic Policy Gradients in Continuous Control with Deep Reinforcement Learning using the Beta Distribution”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 834–843.

[35] Sanjiban Choudhury, Sankalp Arora, and Sebastian Scherer. “The Planner Ensemble and Trajectory Executive: A High Performance Motion Planning System with Guaranteed Safety”. In: *AHS 70th Annual Forum, Montreal, Quebec, Canada*. Vol. 1. 2. 2014, pp. 3–1.

[36] Christopher Choy, JunYoung Gwak, and Silvio Savarese. “4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3075–3084.

[37] Adam Coates and Andrew Y Ng. “Learning feature representations with k-means”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 561–580.

[38] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 3213–3223.

[39] Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. "Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning". In: (2012).

[40] Jin Q. Cui, Swee King Phang, Kevin Z. Y. Ang, Fei Wang, Xiangxu Dong, Yijie Ke, Shupeng Lai, Kun Li, Xiang Li, Feng Lin, Jing Lin, Peidong Liu, Tao Pang, Biao Wang, Kangli Wang, Zhaolin Yang, and Ben M. Chen. "Drones for cooperative search and rescue in post-disaster situation". In: *RAM/CIS*. 2015.

[41] Brian Curless and Marc Levoy. "A volumetric method for building complex models from range images". In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM. 1996, pp. 303–312.

[42] Angela Dai and Matthias Nießner. "3DMV: Joint 3D-Multi-View Prediction for 3D Semantic Scene Segmentation". In: *European Conference on Computer Vision (ECCV)*. 2018.

[43] Carol Deering and Jason Stoker. "Let's agree on the casing of lidar". In: *Lidar Magazine* 4 (Jan. 2014), pp. 48–51.

[44] Jeffrey Delmerico, Alessandro Giusti, Elias Mueggler, Luca Maria Gambardella, and Davide Scaramuzza. ""On-the-spot Training" for Terrain Classification in Autonomous Air-Ground Collaborative Teams". In: *ISER*. 2016.

[45] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, 2009, pp. 248–255.

[46] Daniel C Dennett. *Why and how does consciousness seem the way it seems?* Open MIND. Frankfurt am Main: MIND Group, 2014.

[47] Vishnu R Desaraju, Nathan Michael, Martin Humenberger, Roland Brockers, Stephan Weiss, Jeremy Nash, and Larry Matthies. "Vision-based landing site evaluation and informed optimal trajectory generation toward autonomous rooftop landing". In: *Autonomous Robots* (2015), pp. 1–19.

[48] Jean-Emmanuel Deschaud, David Prasser, M. Freddie Dias, Brett Browning, and Peter Rander. "Automatic data driven vegetation modeling for lidar simulation". In: *ICRA*. 2012, pp. 5030–5036.

[49] Mark De Deuge, Alastair Quadros, Calvin Hung, and Bertrand Douillard. "Unsupervised feature learning for classification of outdoor 3D scans". In: *Australasian Conference on Robotics and Automation (ACRA)*. Vol. 2. 2013, p. 1.

[50] Geetesh Dubey, Ratnesh Madaan, and Sebastian Scherer. "DROAN-Disparity-Space Representation for Obstacle Avoidance: Enabling Wire Mapping & Avoidance". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 6311–6318.

[51] David Eigen and Rob Fergus. “Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-scale Convolutional Architecture”. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 2650–2658.

[52] Claus Emmeche. “Does a robot have an Umwelt? - Reflections on the qualitative biosemiotics of Jakob von Uexküll”. In: *Semiotica* 134 (Nov. 2001).

[53] Jakob Engel, T Schöps, and Daniel Cremers. “LSD-SLAM: Large-scale direct monocular SLAM”. In: *ECCV 2014* (2014), pp. 834–849.

[54] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. “Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1355–1361.

[55] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*.

[56] C. A. F. Ezequiel, M. Cua, N. C. Libatique, G. L. Tangonan, R. Alampay, R. T. Labuguen, C. M. Favila, J. L. E. Honrado, V. Caños, C. Devaney, A. B. Loreto, J. Bacusmo, and B. Palma. “UAV aerial imaging applications for post-disaster assessment, environmental management and infrastructure development”. In: *ICUAS*. 2014, pp. 274–283.

[57] Matthias Faessler, Flavio Fontana, Christian Forster, Elias Mueggler, Matia Pizzoli, and Davide Scaramuzza. “Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor Micro Aerial Vehicle”. In: *Journal of Field Robotics* (2015).

[58] Péter Fankhauser, Michael Bloesch, Christian Gehring, Marco Hutter, and Roland Siegwart. “Robot-Centric Elevation Mapping with Uncertainty Estimates”. In: *International Conference on Climbing and Walking Robots (CLAWAR)*. 2014.

[59] Clément Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. “Scene parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers”. In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012.

[60] Gregory T. Flitton, Toby P. Breckon, and Najla Megherbi. “A 3D extension to cortex like mechanisms for 3D object class recognition”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*. IEEE Computer Society, 2012, pp. 3634–3641.

[61] C. Forster, M. Faessler, F. Fontana, M. Werlberger, and D. Scaramuzza. “Continuous on-board monocular-vision-based elevation mapping applied to autonomous landing of micro aerial vehicles”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 111–118.

[62] Andrea Frome, Daniel Huber, and Ravi Kolluri. “Recognizing objects in range data using regional point descriptors”. In: *Computer Vision-ECCV ...* 1 (2004), pp. 1–14.

[63] Takahiro Fujimori and Takeo Kanade. “An Approach to Knowledge-Based Interpretation of Outdoor Natural Color Road Scenes”. In: *Vision and Navigation: The Carnegie Mellon Navlab*. 1990, pp. 39–81.

[64] Kunihiko Fukushima. “Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position”. In: *Biological Cybernetics* 36 (1980), pp. 193–202.

[65] Cipriano Galindo, Alessandro Saffiotti, Silvia Coradeschi, Pär Buschka, Juan-Antonio Fernandez-Madrigal, and Javier González. “Multi-hierarchical semantic maps for mobile robotics”. In: *2005 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, pp. 2278–2283.

[66] Liuhao Ge, Hui Liang, Junsong Yuan, and Daniel Thalmann. “3D Convolutional Neural Networks for Efficient and Robust Hand Pose Estimation from Single Depth Images”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 5679–5688.

[67] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 2014, pp. 580–587.

[68] A. Giusti, J. Guzzi, D. C. Cireşan, F. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella. “A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots”. In: *IEEE Robotics and Automation Letters* 1.2 (2016), pp. 661–667.

[69] Aleksey Golovinskiy, Vladimir G. Kim, and Thomas A. Funkhouser. “Shape-based recognition of 3D point clouds in urban environments”. In: *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*. IEEE Computer Society, 2009, pp. 2154–2161.

[70] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[71] Ben Graham. “Sparse 3D convolutional neural networks”. In: *Proceedings of the British Machine Vision Conference (BMVC)*. Ed. by Xianghua Xie, Mark W. Jones, and Gary K. L. Tam. BMVA Press, Sept. 2015, pp. 150.1–150.9.

[72] Paul Groves. *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems, Second Edition*. Mar. 2013.

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

[74] Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik. “Learning rich features from RGB-D images for object detection and segmentation”. In: *European Conference on Computer Vision (ECCV)*. Springer. 2014, pp. 345–360.

[75] Raia Hadsell, J Andrew Bagnell, Daniel Huber, and Martial Hebert. “Non-Stationary Space-Carving Kernels for Accurate Rough Terrain Estimation”. In: *IJRR* 29.8 (2010), pp. 981–996.

[76] Raia Hadsell, J. Andrew Bagnell, Daniel Huber, and Martial Hebert. “Space-carving Kernels for Accurate Rough Terrain Estimation”. In: *The International Journal of Robotics Research* 29.8 (2010), pp. 981–996.

[77] D. Hähnel, D. Schulz, and W. Burgard. “Map Building with Mobile Robots in Populated Environments”. In: *IROS*. 2002.

[78] Bharath Hariharan, Pablo Andrés Arbeláez, Ross B. Girshick, and Jitendra Malik. “Hypercolumns for object segmentation and fine-grained localization”. In: *IEEE Conference*

on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015. IEEE Computer Society, 2015, pp. 447–456.

[79] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2980–2988.

[80] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778.

[81] *Helipad Marking And Lighting (STANAG 3619)*. 4th ed. 2007.

[82] Leonard R. Herrmann. "Laplacian-Isoparametric Grid Generation Scheme". In: *Journal of the Engineering Mechanics Division* 102.5 (1976), pp. 749–756.

[83] Hanzhang Hu, Daniel Munoz, J. Andrew Bagnell, and Martial Hebert. "Efficient 3-D scene analysis from streaming data". In: *ICRA* (2013).

[84] Jing Huang and Suya You. "Point cloud labeling using 3D Convolutional Neural Network". In: *International Conference on Pattern Recognition (ICPR)*. 2016, pp. 2670–2675.

[85] L. D. Jackel, Eric Krotkov, Michael Perschbacher, Jim Pippine, and Chad Sullivan. "The DARPA LAGR program: Goals, challenges, methodology, and phase I results". In: *Journal of Field Robotics* 23.11-12 (2006), pp. 945–973.

[86] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. "3D Convolutional Neural Networks for Human Action Recognition". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*. Ed. by Johannes Fürnkranz and Thorsten Joachims. Omnipress, 2010, pp. 495–502.

[87] Andrew Johnson, James Collier, and Aron Wolf. "Lidar-based hazard avoidance for safe landing on Mars". In: *AAS/AIAA Space Flight Mechanics Meeting*. 818. 2001, pp. 1091–1099.

[88] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Fei-Fei Li. "Large-Scale Video Classification with Convolutional Neural Networks". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 2014, pp. 1725–1732.

[89] Alonzo Kelly. "Mobile robotics: Mathematics, models, and methods". In: *Mobile Robotics: Mathematics, Models, and Methods* (Jan. 2013).

[90] Alonzo Kelly, Anthony Stentz, Omead Amidi, Mike Bode, David Bradley, Antonio Diaz-Calderon, Mike Happold, Herman Herman, Robert Mandelbaum, Tom Pilarski, et al. "Toward reliable off road autonomous vehicles operating in challenging environments". In: *The International Journal of Robotics Research* 25.5-6 (2006), pp. 449–483.

[91] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. "End-To-End Learning of Geometry and Context for Deep Stereo Regression". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[92] Yasir Niaz Khan, Andreas Masselli, and Andreas Zell. "Visual terrain classification by flying robots". In: *2012 IEEE International Conference on Robotics and Automation* (2012), pp. 498–503.

[93] Eunyoung Kim and Gérard Medioni. “Urban scene understanding from aerial and ground LIDAR data”. In: *Machine Vision and Applications* 22.4 (2010), pp. 691–703.

[94] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollar. “Panoptic Segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Apr. 2019.

[95] Kurt Konolige, James Bowman, JD Chen, Patrick Mihelich, Michael Calonder, Vincent Lepetit, and Pascal Fua. “View-based maps”. In: *The International Journal of Robotics Research* 29.8 (2010), pp. 941–957.

[96] Ioannis Kostavelis and Antonios Gasteratos. “Semantic mapping for mobile robotics tasks: A survey”. In: *Robotics and Autonomous Systems* 66 (2015), pp. 86–103.

[97] Philipp Krähenbühl and Vladlen Koltun. “Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials”. In: *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*. Ed. by John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger. 2011, pp. 109–117.

[98] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. Ed. by Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger. 2012, pp. 1106–1114.

[99] Benjamin Kuipers. “The spatial semantic hierarchy”. In: *Artificial Intelligence* 119 (2000), pp. 191–233.

[100] Abhijit Kundu, Yin Li, Frank Dellaert, Fuxin Li, and James M Rehg. “Joint Semantic Segmentation and 3D Reconstruction from Monocular Video”. In: *ECCV*. 2014, pp. 1–16.

[101] Kevin Lai, Liefeng Bo, and Dieter Fox. “Unsupervised Feature Learning for 3D Scene Labeling”. In: (2014), pp. 3050–3057.

[102] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. “A large-scale hierarchical multi-view RGB-D object dataset”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 1817–1824.

[103] Jean-François Lalonde, Nicolas Vandapel, Daniel F. Huber, and Martial Hebert. “Natural terrain classification using three-dimensional ladar data for ground robot mobility”. In: *Journal of Field Robotics* 23.10 (2006), pp. 839–861.

[104] Dagmar Lang and Dietrich Paulus. “Semantic maps for robotics”. In: *Proceedings of the Workshop on AI Robotics at ICRA, Chicago, IL, USA*, pp. 14–18.

[105] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[106] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521.7553 (2015), pp. 436–444.

[107] Ian Lenz, Honglak Lee, and Ashutosh Saxena. “Deep Learning for Detecting Robotic Grasps”. In: *International Journal of Robotics Research* 34 (Jan. 2013).

[108] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. “End-to-end training of deep visuomotor policies”. In: *Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[109] Guosheng Lin, Chunhua Shen, Anton van den Hengel, and Ian Reid. “Exploring Context with Deep Structured models for Semantic Segmentation”. In: (2016), pp. 1–14. arXiv: 1603.03183.

[110] Guosheng Lin, Chunhua Shen, Anton van den Hengel, and Ian D. Reid. “Efficient Piecewise Training of Deep Structured Models for Semantic Segmentation”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 3194–3203.

[111] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. “Microsoft COCO: Common objects in context”. In: *Computer Vision-ECCV 2014*. Springer, 2014, pp. 740–755.

[112] Ming-Yu Liu, Shuoxin Lin, Srikumar Ramalingam, and Oncel Tuzel. “Layered Interpretation of Street View Images”. In: *CoRR* abs/1506.04723 (2015).

[113] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 3431–3440.

[114] Wenjie Luo, Alexander G. Schwing, and Raquel Urtasun. “Efficient Deep Learning for Stereo Matching”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 5695–5703.

[115] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. “Rectifier nonlinearities improve neural network acoustic models”. In: *ICML*. Vol. 30. 2013.

[116] Jose Macedo, Roberto Manduchi, and Larry Matthies. “Ladar-based discrimination of grass from obstacles for autonomous navigation”. In: *ISER*. 2001.

[117] Ratnesh Madaan, Daniel Maturana, and Sebastian Scherer. “Wire detection using synthetic data and dilated convolutional networks for unmanned aerial vehicles”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3487–3494.

[118] Roberto Manduchi, Andres Castano, Ashit Talukder, and Larry Matthies. “Obstacle detection and terrain classification for autonomous off-road navigation”. In: *Autonomous robots* 18.1 (2005), pp. 81–102.

[119] L Matthies, C Bergh, A Castano, and J Macedo. “Obstacle detection in foliage with ladar and radar”. In: *ISRR*. 2003.

[120] D. Maturana and S. Scherer. “3D Convolutional Neural Networks for Landing Zone Detection from LiDAR”. In: *ICRA*. 2015.

[121] Xuelian Meng, Nate Currit, and Kaiguang Zhao. “Ground Filtering Algorithms for Airborne LiDAR Data: A Review of Critical Issues”. In: *Remote Sensing* 2.3 (2010), pp. 833–860.

[122] H. Moravec and A Elfes. “High resolution maps from wide angle sonar”. In: *ICRA*. 1985.

[123] Mohammadreza Mostajabi, Payman Yadollahpour, and Gregory Shakhnarovich. “Feedforward semantic segmentation with zoom-out features”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 3376–3385.

[124] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan L. Yuille. “The Role of Context for Object Detection and Semantic Segmentation in the Wild”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 2014, pp. 891–898.

[125] D. Munoz, N. Vandapel, and M. Hebert. “Onboard contextual classification of 3-D point clouds with learned high-order Markov Random Fields”. In: *2009 IEEE International Conference on Robotics and Automation* (2009), pp. 2009–2016.

[126] Daniel Munoz, James A. Bagnell, Nicolas Vandapel, and Martial Hebert. “Contextual classification with functional Max-Margin Markov Networks”. In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, 2009, pp. 975–982.

[127] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. “ORB-SLAM: a Versatile and Accurate Monocular SLAM System”. In: *IEEE Transactions on Robotics* (2015), p. 15. arXiv: 1502.00956.

[128] Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. “OpenVDB: An Open-source Data Structure and Toolkit for High-resolution Volumes”. In: *ACM SIGGRAPH 2013 Courses*. SIGGRAPH ’13. ACM, 2013, 19:1–19:1.

[129] Pushmeet Kohli Nathan Silberman Derek Hoiem and Rob Fergus. “Indoor Segmentation and Support Inference from RGBD Images”. In: *European Conference on Computer Vision (ECCV)*. 2012.

[130] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. “KinectFusion: Real-Time Dense Surface Mapping and Tracking”. In: *IEEE ISMAR*. IEEE, 2011.

[131] D. Nister, O. Naroditsky, and J. Bergen. “Visual odometry”. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 1. 2004, pp. I–I.

[132] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. “Learning Deconvolution Network for Semantic Segmentation”. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 1520–1528.

[133] Andreas Nüchter and Joachim Hertzberg. “Towards semantic maps for mobile robots”. In: *Robotics and Autonomous Systems* 56.11 (), pp. 915–926.

[134] Simon Ottenhaus, Daniel Renninghoff, Raphael Grimm, Fabio Ferreira, and Tamim Asfour. “Visuo-Haptic Grasping of Unknown Objects based on Gaussian Process Implicit Surfaces and Deep Learning”. In: *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*. 2019, pp. 402–409.

[135] James Paduano, John Wissler, Graham Drozeski, Michael Piedmonte, Navid Dadkhah, Jim Francis, Charles Shortlidge, Jason Bold, Fritz Langford, Mark Chaoui, Cheng-Jen Liu, Eric Foster, Sanjiv Singh, Lyle Chamberlain, Brad Hamner, Hugh Cover, Adam Stambler, Ayman Singh, Samuel Nalbone, Marcel Bergerman, Sebastian Scherer, Sanjiban Choudhury, Silvio Maeta, Sankalp Arora, Daniel Maturana, Cindy Dominguez, Brian Moon, Robert Strouse, Lisa Papautsky, David Mindell, Dino Cerchie, Bryan Chu, Jason Graham, Christine Cameron, Mark Hardesty, Roger Hehr, Doug Limbaugh, James Bona, David Barnhard, and Desare'a Chessar. "TALOS: An Unmanned Cargo-Delivery System for Rotorcraft Landing to Unprepared Sites". In: *AHS*.

[136] Stephen E. Palmer. *Vision science: photons to phenomenology*. Cambridge, Mass.: MIT Press, 1999, p. 810.

[137] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. "ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation". In: *CoRR* abs/1606.02147 (2016).

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

[139] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. "Pointnet: Deep learning on point sets for 3D classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[140] Xiaojuan Qi, Renjie Liao, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. "3D Graph Neural Networks for RGBD Semantic Segmentation". In: *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, pp. 5209–5218.

[141] Tong Qin, Peiliang Li, and Shaojie Shen. "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator". In: *IEEE Transactions on Robotics* 34.4 (Aug. 2018), pp. 1004–1020.

[142] A. Quadros, James Patrick Underwood, and Bertrand Douillard. "An occlusion-aware feature for range images". In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4428–4435.

[143] S. Razakarivony and F. Jurie. *Technical Report GREYC-2015-03-04*. 2015.

[144] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. 2016.

[145] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 779–788.

[146] Vladimir Reilly, Haroon Idrees, and Mubarak Shah. "Detection and tracking of large number of targets in wide area surveillance". In: *LNCS* 6313 LNCS (2010), pp. 186–199.

[147] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by Corinna Cortes,

Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett. 2015, pp. 91–99.

[148] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett. 2015, pp. 91–99.

[149] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. "OctNet: Learning Deep 3D Representations at High Resolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.

[150] Eleanor Rosch. "Natural Categories". In: *Cognitive Psychology* 4.3 (1973).

[151] Havard Rue and Leonhard Held. *Gaussian Markov random fields: theory and applications*. CRC press, 2005.

[152] Sebastian Scherer. "Low-altitude operation of unmanned rotorcraft". PhD thesis. Carnegie Mellon University, 2011.

[153] Sebastian Scherer, Lyle Chamberlain, and Sanjiv Singh. "First results in autonomous landing and obstacle avoidance by a full-scale helicopter". In: *2012 IEEE International Conference on Robotics and Automation* (2012), pp. 951–956.

[154] Sebastian Scherer, Lyle Chamberlain, and Sanjiv Singh. "Online assessment of landing sites". In: *AIAA Infotech@ Aerospace 2010*. 2010, p. 3358.

[155] Max Schwarz, Hannes Schulz, and Sven Behnke. "RGB-D Object Recognition and Pose Estimation based on Pre-trained Convolutional Neural Network Features". In: *IEEE International Conference on Robotics and Automation (ICRA)* May (2015).

[156] John Secord and Avideh Zakhor. "Tree detection in urban regions using aerial lidar and image data". In: *IEEE Geoscience and Remote Sensing Letters* 4.2 (2007), pp. 196–200.

[157] Sunando Sengupta, Paul Sturgess, L'ubor Ladicky, and Philip H. S. Torr. "Automatic dense visual semantic mapping from street-level imagery". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2012), pp. 857–862.

[158] R Shapovalov, a Velizhev, O Barinova, and a Konushin. "Directional associative markov network for 3-D point cloud classification". In: *PCV*. 2010.

[159] Roman Shapovalov and Alexander Velizhev. "Cutting-Plane Training of Non-associative Markov Network for 3D Point Cloud Segmentation". In: *2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission* (2011), pp. 1–8.

[160] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps". In: *CoRR* (2013), pp. 1–8. arXiv: 1312.6034.

[161] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[162] George Sithole and George Vosselman. “Experimental comparison of filter algorithms for bare-Earth extraction from airborne laser scanning point clouds”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 59.1 (2004). Advanced Techniques for Analysis of Geo-spatial Data, pp. 85–101.

[163] Richard Socher, Brody Huval, Bharath Putta Bath, Christopher D. Manning, and Andrew Y. Ng. “Convolutional-Recursive Deep Learning for 3D Object Classification”. In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. Ed. by Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger. 2012, pp. 665–673.

[164] Yu Song, Stephen Nuske, and Sebastian Scherer. “A multi-sensor fusion MAV state estimation from long-range stereo, IMU, GPS and barometric sensors”. In: *Sensors* 17.1 (2017), p. 11.

[165] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. “Multi-view convolutional neural networks for 3D shape recognition”. In: *Proc. ICCV*. 2015.

[166] L. J. Sugarbaker, D. F. Eldridge, A. L. Jason, Vicki Lukas, D. L. Saghy, J. M. Stoker, and D. R. Thunen. *Status of the 3D Elevation Program, 2015: U.S. Geological Survey Open-File Report*. Tech. rep. 2017.

[167] Christian Szegedy, Scott Reed, Dumitru Erhan, and Dragomir Anguelov. “Scalable, High-Quality Object Detection”. In: *arXiv preprint arXiv:1412.1441* (2014).

[168] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 1st. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

[169] Sebastian Thrun. “Learning Occupancy Grid Maps with Forward Sensor Models”. In: *Autonomous Robots* 15.2 (2003), pp. 111–127.

[170] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[171] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. “Stanley: The robot that won the DARPA Grand Challenge”. In: *Journal of Field Robotics* 23.9 (2006), pp. 661–692.

[172] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. “Stanley: The Robot That Won the DARPA Grand Challenge: Research Articles”. In: *J. Robot. Syst.* 23.9 (2006), pp. 661–692.

[173] G. D. Tipaldi, L. Spinello, and W. Burgard. “Geometrical FLIRT phrases for large scale place recognition in 2D range data”. In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 2693–2698.

[174] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. "Selective search for object recognition". In: *International journal of computer vision* 104.2 (2013), pp. 154–171.

[175] Chris Urmson, Charlie Ragusa, David Ray, Joshua Anhalt, Daniel Bartz, Tugrul Galatali, Alexander Gutierrez, Josh Johnston, Sam Harbaugh, Hiroki "Yu" Kato, William Messner, Nick Miller, Kevin Peterson, Bryon Smith, Jarrod Snider, Spencer Spiker, Jason Ziglar, William "Red" Whittaker, Michael Clark, Phillip Koon, Aaron Mosher, and Josh Struble. "A robust approach to high-speed navigation for unrehearsed desert terrain". In: *Journal of Field Robotics* 23.8 (2006), pp. 467–508.

[176] Christopher Urmson, Joshua Anhalt, Hong Bae, J. Andrew (Drew) Bagnell, Christopher R. Baker, Robert E Bittner, Thomas Brown, M. N. Clark, Michael Darms, Daniel Demitrish, John M Dolan, David Duggins, David Ferguson, Tugrul Galatali, Christopher M Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas Howard, Sascha Kolski, Maxim Likhachev, Bakhtiar Litkouhi, Alonzo Kelly, Matthew McNaughton, Nick Miller, Jim Nickolaou, Kevin Peterson, Brian Pilnick, Ragunathan Rajkumar, Paul Rybski, Varsha Sadekar, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod M Snider, Joshua C Struble, Anthony (Tony) Stentz, Michael Taylor, William (Red) L. Whittaker, Ziv Wolkowicki, Wende Zhang, and Jason Ziglar. "Autonomous driving in urban environments: Boss and the Urban Challenge". In: *JFR* 25.8 (2008). Ed. by Sanjiv Singh Martin Buehler, Karl Lagnemma, pp. 425–466.

[177] Abhinav Valada, Gabriel L Oliveira, Thomas Brox, and Wolfram Burgard. "Deep Multispectral Semantic Scene Understanding of Forested Environments using Multimodal Fusion". In: *ISER*. 2016.

[178] Shrihari Vasudevan and Roland Siegwart. "Bayesian space conceptualization and place classification for semantic maps in mobile robotics". In: *Robotics and Autonomous Systems* 56.6 (), pp. 522–537.

[179] Vibhav Vineet, Ondrej Miksik, Morten Lidegaard, Matthias Nießner, Stuart Golodetz, Victor A. Prisacariu, Olaf Kähler, David W. Murray, Shahram Izadi, Patrick Perez, and Philip H. S. Torr. "Incremental Dense Semantic Stereo Fusion for Large-Scale Semantic Scene Reconstruction". In: *ICRA*. 2015, pp. 75–82.

[180] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. "Dynamic Graph CNN for Learning on Point Clouds". In: *ACM Trans. Graph.* 38.5 (2019).

[181] Carl Wellington. "Learning a Terrain Model for Autonomous Navigation in Rough Terrain". In: *Change* (2005).

[182] David S. Wettergreen and Michael D. Wagner. "Developing a framework for reliable autonomous surface mobility". In: *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)* 1 (2012).

[183] M Whalley, M Takahashi, J Fletcher, E Moralez, C Ott, M Olmstead, C Goerzen, G Schulein, J Savage, and H Burns. "Flight Test Results for Autonomous Obstacle Field Navigation and Landing Site Selection on the RASCAL JUH-60A". In: *69th Annual Forum of the American Helicopter Society*. Phoenix, AZ, 2012.

[184] Zhirong Wu, Shuran Song, Aditya Khosla, Xiaoou Tang, and Jianxiong Xiao. “3D ShapeNets for 2.5D Object Recognition and Next-Best-View Prediction”. In: *CoRR* abs/1406.5670 (2014).

[185] Xuehan Xiong, Daniel Munoz, J. Andrew Bagnell, and Martial Hebert. “3-D scene analysis via sequenced predictions over points and regions”. In: *2011 IEEE International Conference on Robotics and Automation* (2011), pp. 2609–2616.

[186] Wai Yeung Yan, Ahmed Shaker, and Nagwa El-Ashmawy. “Urban land cover classification using airborne LiDAR data: A review”. In: *Remote Sensing of Environment* 158 (2015), pp. 295–310.

[187] Luke Yoder and Sebastian Scherer. “Autonomous Exploration for Infrastructure Modeling with a Micro Aerial Vehicle”. In: *FSR 2015*. Springer. 2015, pp. 427–440.

[188] Fisher Yu and Vladlen Koltun. “Multi-Scale Context Aggregation by Dilated Convolutions”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016.

[189] Ji Zhang and Sanjiv Singh. “LOAM: Lidar Odometry and Mapping in Real-time”. In: *Robotics: Science and Systems Conference (RSS 2014)*. 2014.

[190] Bolei Zhou, Àgata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. “Learning Deep Features for Scene Recognition using Places Database”. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger. 2014, pp. 487–495.

[191] Shengyan Zhou, Junqiang Xi, Matthew W McDaniel, Takayuki Nishihata, Phil Salesses, and Karl Iagnemma. “Self-supervised learning to visually detect terrain surfaces for autonomous robots operating in forested terrain”. In: *Journal of Field Robotics* 29.2 (2012), pp. 277–297.

[192] Yukun Zhu, Raquel Urtasun, Ruslan Salakhutdinov, and Sanja Fidler. “segDeepM: Exploiting segmentation and context in deep neural networks for object detection”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 4703–4711.

[193] Michael Zollhöfer. “Commodity RGB-D Sensors: Data Acquisition”. In: *RGB-D Image Analysis and Processing*. Springer International Publishing, 2019.