Imperial College London
Department of Computing

SLAM and Deep Learning for 3D Indoor Scene Understanding

John B. McCormac

December 2018

Supervised by Professor Andrew Davison
Co-supervised by Dr Stefan Leutenegger

Abstract

We build upon research in the fields of Simultaneous Localisation and Mapping (SLAM) and Deep Learning to develop 3D maps of indoor scenes that not only describe *where* things are but *what* they are. We focus on real-time online methods suitable for applications such as domestic robotics and augmented reality. While early approaches to SLAM used sparse feature maps for localisation, recent years have seen the advent of real-time dense SLAM systems which enabled applications not possible with only sparse feature maps. Further augmenting dense maps with semantic information will in future enable more intelligent domestic robots and more intuitive human-map interactions not possible with map geometry alone.

Early work presented here sought to combine recent advances in semantic segmentation using Convolutional Neural Networks (CNNs) with dense SLAM approaches to produce a semantically annotated dense 3D map. Although we found this combination improved segmentation performance, its inherent limitations subsequently led to a paradigm shift away from semantic annotation towards instance detection and 3D object-level mapping. We propose a new type of SLAM system consisting of discovered object instances that are reconstructed online in individual volumes. We develop a new approach to robustly combine multiple associated 2D instance mask detections into a fused 3D foreground segmentation for each object. The use of individual volumes allows the relative poses of objects to be optimised in a pose-graph, producing a consistent global map that allows objects to be reused on loopy trajectories, and which can improve reconstruction quality.

A notable feature of CNNs is their ability to make use of large annotated datasets, and so we also explore methods to reduce the cost of indoor semantic dataset production. We explore SLAM as a means of mitigating labour intensive annotation of video data, but found that producing a large-scale dataset with such an approach would still require significant resources. We therefore explore automated methods to produce a large-scale photorealistic synthetic dataset of indoor trajectories at low cost, and we verify the benefits of the dataset on the task of semantic segmentation. To automate trajectory generation we present a novel two-body random trajectory method that mitigates issues of a completely random approach, and which has subsequently been used in other synthetic indoor datasets.

Acknowledgements

This work would not have been possible without the support and encouragement of many people who have helped to shape both me and my research.

First, I cannot thank my supervisors enough. Professor Andrew Davison and Dr Stefan Leutenegger have both been enormously generous with their time and expertise. They have taught me a great deal and trusted me when I wanted to explore this new 'deep learning thing.' Andy's steady hand of vision and insightful wisdom has shaped the entire trajectory of my research. Stefan's guidance has been no less instrumental, and I particularly appreciate his patience during the long hours at the whiteboard together.

I have had the good fortune to work with and beside many extremely talented researchers. I thank Ankur Handa and Ronnie Clark for all of their help and enjoyable company during late nights and through paper deadlines. Patrick Bardow for his companionship and mathematical and programming knowledge as we worked through the challenges of a PhD together. Michael Bloesch who also spent hours teaching me at the whiteboard. Robert Lukierski who could fix almost anything and did so frequently. Jacek Zienkiewicz for his great advice and constant confidence. Wenbin Li and Sajad Saeedi for pushing forward when I could not. Iosifina Pournara who helped to organise everything and keep me right. Tom Whelan and Renato Salas-Moreno for help at the very start of my PhD. I have received helpful advice from and had fruitful discussions with many other lab members including Ed Johns, Hanme Kim, Jan Czarnowski, Tristan Laidlow, Andrea Nicastro, Dimos Tzoumanikas, Jan Jachnik, Lukas Platinsky and everyone else in the lab as well.

I am grateful to Archeeta for her patience, support, and kindness through the trials and tribulations of PhD life. Our travels have been a source of joy even in the most stressful times. Thank you.

I thank my family. Their unconditional confidence and support has provided me with a foundation that very few are blessed with, and I am so grateful to be one of them.

I also thank Dyson Technology Ltd. for funding this research.

Contents

Chapter 1

Introduction

Contents

A single glance around a room is enough for a person to achieve an incredible level of understanding of their environment. They know where they are in relation to the doorway and walls, they can see the free path through obstacles on the floor, and they know what the objects on the table are and where to tidy them. We understand the visual environment so well and intuitively that it is easy to forget just how complicated a process it truly is. Famously, computer vision began as an MIT summer project in 1966[1] during which time they aimed to construct a significant part of a visual system by performing figure-ground segmentation and object identification.

[1]The original memo is still available: http://hdl.handle.net/1721.1/6125.

Over 50 years of research has passed and it has led to huge improvements in computer vision algorithms and the capabilities of the computers that run them. The application of Deep Learning to some of these problems has allowed machines to reach or even exceed human-level performance in certain specific visual tasks such as fine-grained classification [He et al., 2015, Russakovsky et al., 2015] (although 'human-level' may of course vary from person to person). Similar approaches to pattern recognition form part of the system which allowed computers to beat expert humans at the historically insurmountable game of Go [Silver et al., 2016]. These results are impressive, but replicating the robustness and generality of human vision is a challenge that has not yet been achieved, and much more research is still needed before computers will finally glance around a scene and properly understand it.

To know your place in an environment requires knowledge of your location as well as some form of understanding or map of the scene itself. Maps serve a dual purpose, they allow more accurate localisation than dead-reckoning (which quickly accumulates drift) and they assist in the operation of tasks within the map such as path planning. Methods developed for this problem domain, known as Simultaneous Localisation And Mapping (SLAM), are used heavily in this thesis. The focus will be on processing input from a camera and so to be precise our problem domain is known as Visual SLAM. Visual SLAM is commonly associated with a passive monocular camera, but here we make use of readily available commodity depth (RGB-D) cameras which can provide a video stream of dense range measurements registered to normal RGB frames. In practical applications additional input modalities such as Inertial Measurement Units (IMUs) can also be incorporated to improve the robustness and accuracy of the system [Laidlow et al., 2017], however that has not been explored in this work.

The research in this thesis aims to explore some of the ways in which the recent advances in Deep Learning, and more specifically Convolutional Neural Networks (CNNs), can be combined with SLAM to automatically produce 3D maps of indoor scenes that include both geometric and semantic elements. Including semantic information in a map is a necessity for enabling domestic robotic agents to advance beyond obstacle avoidance and it can also enable more intuitive human-map interactions. Sharing a common spatial and semantic map allows intricate commands such as 'Fetch that coffee mug from the table beside you' to be properly interpreted. Similarly, the ability to query semantic information within a map is useful for humans directly, providing a convenient database for storing and answering queries about a previously mapped environment; 'How many chairs do we have in the conference room?' In short we are concerned with knowing *where* things are and *what* they are.

In this thesis we begin with research into semantic annotation of dense 3D indoor scenes. Following the 'map everything' mantra of dense reconstruction we aim to annotate everything in a manner not unlike painting a texture on the geometry itself. Research in this area led to the development of a system called 'SemanticFusion' [McCormac et al., 2017a] shown on the left of Figure 1.1. We used the dense SLAM system ElasticFusion [Whelan et al., 2015b] to provide a 3D map into which we could fuse 2D semantic predictions from a CNN. We found ElasticFusion to be a particularly suitable foundation for producing a real-time dense 3D semantic map. Its surfel-based map representation could conveniently store semantic probabilities and its map deformation approach to loop closure provided a consistent global map which allowed semantic information to be efficiently fused using a Bayesian update scheme.

In SemanticFusion our experiments showed that a straightforward combination of a dense SLAM system with a CNN could lead to improvements in semantic segmentation accuracy as well as produce an annotated 3D map. However we also found the limits of what could be achieved with the representation. Interpreting and using a dense map with millions of independent semantic probability distributions does not allow task-relevant queries such as 'How many chairs do we have in the conference room?' to be directly answered. The independence of surface patches also allows semantic inconsistencies, such as two halves of the same object being assigned different semantic labels.

To overcome these limitations we explore the new paradigm of object-level mapping; an intuitive map representation which is composed of discovered and reconstructed object instances. This work called Fusion++ [McCormac et al., 2018] was inspired by the SLAM++ system of [Salas-Moreno et al., 2013]. It takes advantage of modern advances in Deep Learning and instance segmentation to mitigate the past limitations of SLAM++ that necessitated a predefined database of reconstructed objects. Instead, in Fusion++, object instances are detected and segmented with a CNN and reconstructed online in individual Truncated Signed Distance Function (TSDF) [Curless and Levoy, 1996] volumes to create an object-level pose-graph (visualised on the right of Figure 1.1). Splitting the map into separate objects and

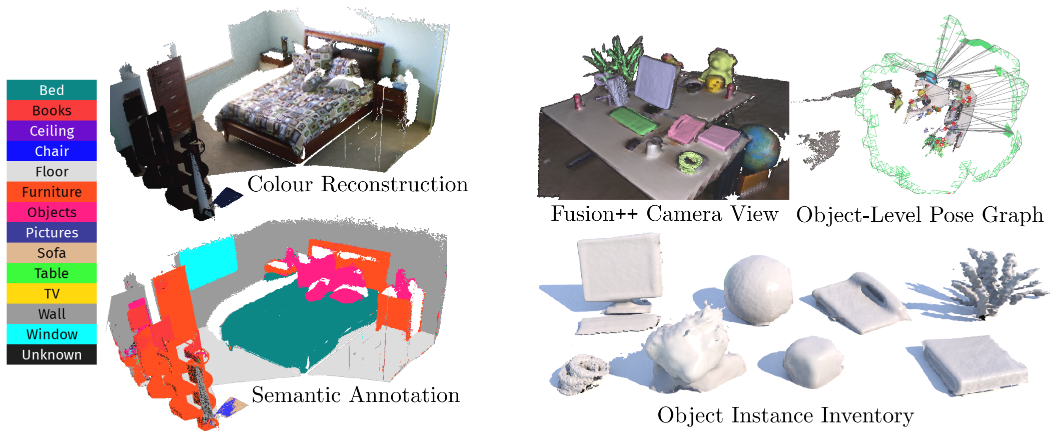


Figure 1.1: **Left:** SemanticFusion [McCormac et al., 2017a] produces a semantically annotated 3D reconstruction of an indoor scene. **Right:** Fusion++ [McCormac et al., 2018] makes a map of individual objects discovered using Deep Learned instance segmentation and volumetrically reconstructed as landmarks in a pose graph.

using a pose-graph not only produces semantically meaningful map entities, but can also improve reconstruction quality by updating the relative poses of objects rather than by eroding and rebuilding their surface geometry on loop-closures.

Our initial approach to reconstructing objects in a volume was to only fuse depth measurements from the foreground of the segmented object instance. Unfortunately this approach was not ideally suited to the variability in instance mask predictions. It discards useful measurements about surface geometry near to the object, which may be needed should the predicted segmentation mask increase in size. It also causes problems when the segmentation mask decreases in size, as previously reconstructed surface geometry persists and would require some other mechanism for removal. Instead we reconstruct all of the geometry within the 3D volume, regardless of the instance mask, and use a fused 3D foreground probability mask to denote the surfaces that belong to the object. We found that fusing the 2D instance masks using a multiplicative Bayesian update scheme, such as that of SemanticFusion, was unstable. A single mask prediction with a very low foreground probability would occasionally delete the fused foreground mask that had been refined over many previous frames. To mitigate this, we propose a more robust alternative which views the binarised mask as the result of a binomial trial, and uses a Beta distribution conjugate prior to calculate the expectation a given voxel corresponds to the object's foreground.

In SLAM++, Iterative Closest Point (ICP) was used to track individual objects in order to produce the edge measurements necessary for an object-level pose-graph. We found that the greater variety of object geometries discovered in Fusion++, as well as the inherent noise from the input depth map during the initial stages of online reconstruction, led ICP to frequently fail to properly converge for individual objects. In order to calculate edge constraints for each object, we therefore developed a modified approach which performed ICP on the combined scene geometry and then optimised the partitioned residual errors for each object instance to calculate the final edge measurement.

Both SLAM and Deep Learning have benefited greatly from ever improving computational hardware. In particular, the advent of readily accessible high performance massively parallel Graphics Processing Units (GPUs) has allowed researchers to exploit the parallel nature of certain problems in both dense SLAM and semantic prediction using CNNs. For many applications from robotics to augmented reality the capacity to operate in real-time is a practical necessity, and in this thesis we make use of GPU hardware to assist in the quest for performance. Although not all of the systems presented here manage to attain true real-time performance, it remains an important focus throughout. The algorithms are designed for online operation and selected for computational efficiency with the potential for real-time use at the very least.

The approach taken here of combining SLAM with Deep Learning is quite modular in nature, using systems and structures that have been studied independently in SLAM and including within them learned methods for semantic and instance predictions from CNNs. This approach is by no means the only approach that can be adopted, and it remains an open question whether the robustness of a fully learned end-to-end SLAM system will eventually prove superior. There has been a recent trend of these two approaches meeting in the middle. As the limits of straightforward CNNs have become clear, a number of recent works have attempted to merge both approaches at a deeper level using domain specific knowledge from both disciplines [Bloesch et al., 2018, Zhou et al., 2018]. The goal is to achieve the best of both worlds, with classically understood SLAM methods being embedded within and between learned units which can improve the robustness where past systems have been particularly fragile.

The importance of large well-annotated domain specific datasets for training and validating CNNs is hard to overestimate. CNNs are able to make good use of very large datasets and it was their performance after being trained on large datasets in a supervised manner that led to the recent surge in their popularity. This thesis also explores the problem of data production for indoor scene understanding. In this context the symbiosis of SLAM and Deep Learning takes on a different form, with SLAM acting as a tool to provide training examples for Deep Learning. The burden of frame-by-frame labelling of video datasets for semantic segmentation can be reduced using a dense 3D reconstruction and the estimated camera trajectory thus assisting in the generation of domain specific training data. The result of this work was the Object Tagger software described in Chapter 3.

Tools such as the Object Tagger cannot entirely mitigate the substantial cost and difficulty of generating real-world datasets. One potential alternative is to use synthetic computer generated datasets which can easily provide a plethora of ground-truth labels. This approach has become quite common in many areas of Deep Learning, with much research on Deep Reinforcement Learning notably focused on playing simple 2D Atari games [Mnih et al., 2015] and more recently moving towards suites of 3D simulated artificial tasks [Brockman et al., 2016]. Synthetic data has also been applied to the problem of SLAM in order to evaluate the trajectory and reconstruction performance of certain methods under varying conditions [Handa et al., 2012, Handa et al., 2014]. Here we apply this approach to our problem domain in order to automate large-scale dataset generation for indoor scene understanding from a moving camera.

To produce such a large-scale dataset with a wide variety of training examples we relied heavily on random sampling. For the scene configuration we sampled objects that were then randomly positioned within the scene and a physics engine was used to produce a physically realistic final configuration. Previous approaches to trajectory generation, such as transplanting hand-captured trajectories into the scene [Handa et al., 2014], would not respect the often complex scene geometry, and also could not produce the variety of trajectories desired. We therefore explored automated methods to produce random indoor trajectories. We found the naive approach of random collision-aware translations and rotations to produce unrealistic trajectories and degenerate training examples such as prolonged panning along plain walls. To alleviate these issues we propose a novel two-body random trajectory method, consisting of a camera position and a look-at point which operates as a

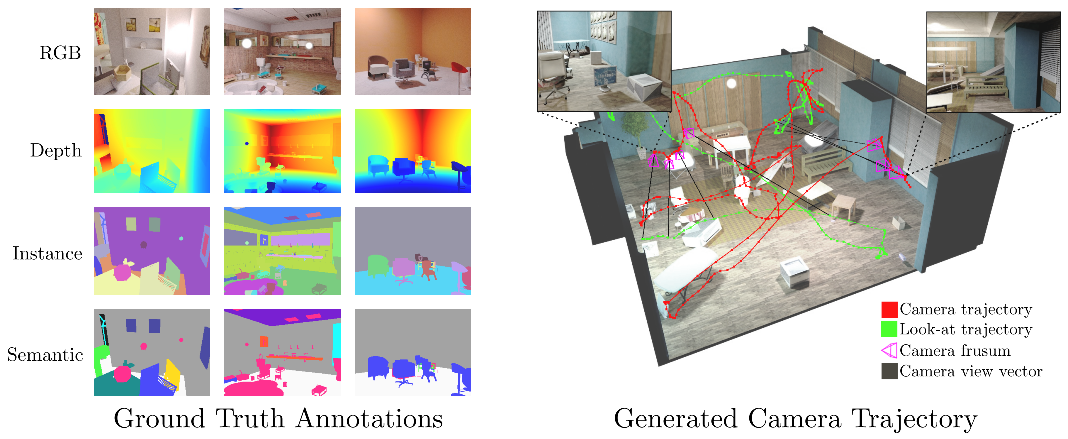


Figure 1.2: **Left:** Photorealistic rendering of indoor scenes with ground truth labels available in the SceneNet RGB-D dataset [McCormac et al., 2017b]. **Right:** A visualisation of the random camera trajectory generated for a synthetic scene.

proxy for a point of focus. This approach has been used as a comparison method in later research on data-driven view point selection [Genova et al., 2017] and has since been refined in a subsequent large-scale synthetic dataset called InteriorNet produced by [Li et al., 2018] on which we collaborated.

The result of the above research was the SceneNet RGB-D dataset [McCormac et al., 2017b] which consists of 5M synthetic images of trajectories in indoor scenes (See Figure 1.2). To verify its usefulness we compared the semantic segmentation performance of network weights produced from pre-training on SceneNet RGB-D with generic ImageNet weights (which lack a task-specific decoder) and found that after fine-tuning, the synthetically pre-trained network outperformed on real-world datasets. The necessity of fine-tuning in order to transfer to real-world data is an important reminder that issues regarding the 'reality-gap' persist, and that the data-set's approximation to the real-world is by no means perfect. However the dataset has continued to be of use to researchers on a variety of related problems [Shamwell et al., 2017, Balloch and Chernova, 2017, Chen and Deng, 2018, Bloesch et al., 2018].

Concurrent and subsequent research has continued to produce work on synthetic indoor environments. The parallel work of [Song et al., 2017] produced the SUN-CG dataset of synthetic scene layouts and the follow-up work of [Zhang et al., 2017] used those layouts to generate physically-based RGB renderings of a still camera within the scenes. This approach differs from our fully-automated random approach by using hand-designed layouts, which has the benefit of contextual information missing from our randomly simulated scene generation pipeline.

A combination of the two approaches, with reasonable manually designed scene layouts or semantic constraints alongside physically simulated randomness, may provide the best of both worlds. This combined approach has started to be explored in the already mentioned InteriorNet dataset [Li et al., 2018]. They use millions of production-level 3D assets and professionally designed indoor layouts to produce a photorealistic dataset at an even larger scale than SceneNet RGB-D. They use the mass and surface material properties of objects in those scenes, as well as a physics engine, to simulate traces of daily life. We believe that in future, as the ability to simulate reality improves and becomes a better approximation of the real-world, so the step from operating in synthetic environments to operating in the real-world will become ever smaller.

As this thesis is concerned with both SLAM and Deep Learning, a brief historical review of both of these strands of research is given below. References to more thorough historical surveys will be given for further reading, as only a general outline of key milestones is given along with more specific discussions on work closely related to the research presented in this thesis.

1.1 SLAM from Sparse to Dense

The work of [Moravec, 1977] on autonomous robotic navigation using Visual Odometry (VO) was one of the earliest predecessors to visual SLAM. In that work the Stanford 'cart' had a single camera mounted on a 50cm rail in order to capture stereo images. These images were used to estimate the 3D location of obstacles using matched features. After the cart moved this process would be repeated and the feature matches used to estimate the current location of the cart. This system was impressively used to automatically navigate a 20m long course while avoiding various obstacles, although it took the robot approximately five hours to do so. This achievement represented a significant advancement towards real-world autonomous navigation and mapping beyond previous work that relied on simple line following [Moravec, 1980]. The envisioned application at that time was a Mars Rover which would require autonomous navigation given the radio transmission delay between the Earth and Mars.

Unlike VO, in Visual SLAM map elements are incrementally refined using repeated measurements from different viewpoints. Revisiting a previously mapped area allows 'loop closures' to remove accumulated drift from the map and enables the agent to self-localise against a global map. As noted by [Matthies et al., 2007], in the context of typical rover navigation SLAM does actually not add a great deal to VO, as rovers tend to travel in one direction rather than revisiting previous landmarks repeatedly. In fact VO continues to be successfully applied to Martian navigation to the present day on the rover Curiosity [Cheng et al., 2005]. The purview of this thesis is the more terrestrial challenges associated with indoor domestic scenes where loopy trajectories dominate and visual SLAM becomes the essential tool to build and maintain an increasingly accurate map.

According to the historical survey of SLAM by [Durrant-Whyte and Bailey, 2006, Bailey and Durrant-Whyte, 2006] the formulation of the modern probabilistic SLAM problem occurred at the 1986 IEEE International Conference on Robotics and Automation. Following work produced the statistical framework to jointly estimate the map landmarks and camera pose from noisy measurements in a probabilistic fashion [Smith and Cheeseman, 1986, Durrant-Whyte, 1988, Smith et al., 1988, Moutarlier and Chatila, 1989]. One of the key observations of this work was that the different landmark measurements were correlated as a result of the uncertainty in the pose of the observer, and by jointly considering all the information all of these parameters could be more accurately estimated. The suggested approach to this problem was an Extended Kalman Filter (EKF). Built upon the Kalman Filter [Kalman, 1960], the EKF is able to work with non-linear transition and measurement functions. In the work of [Leonard and Whyte, 1991] which is widely regarded as the first implementation of a complete SLAM system, an EKF was used to build and refine a map while localising using sonar sensors.

EKFs became a mainstay of SLAM systems in the 1990s [Castellanos, 1998, Davison, 1998, Newman, 1999]. In the recent work of [Cadena et al., 2016] which describes the current state of SLAM research, the EKF has been described as one of the key algorithms in the *classical age* of SLAM, together with Rao-Blackwellized Particle Filters [Montemerlo et al., 2002] and Maximum a Posteriori (MAP) methods. These methods and their history are well covered in the *Probabilistic Robotics* textbook by [Thrun et al., 2005]. In the late 1990s MAP graph-based approaches to SLAM arrived with the work of [Lu and Milios, 1997] and [Gutmann and Konolige, 1999]. These methods are closely related to the field of Structure from Motion (SfM) and the approach called Bundle Adjustment which aims to minimise the reprojection error of 3D points in an image. The difference between SfM and SLAM is that SLAM must be online and incremental [Cadena et al., 2016] and its factors can include a wide range of modalities rather than being limited to projective geometry. Later research by [Strasdat et al., 2012] showed the superiority of graph based approaches over filtering in terms of accuracy and efficiency with modern computational capacity. Pose-graphs have become widely adopted in modern SLAM systems and are indeed used here in Chapter 6.

1.1.1 Feature-based SLAM

The first demonstrated real-time monocular SLAM system called MonoSLAM was presented by [Davison, 2003]. It used an EKF and [Shi and Tomasi, 1994] keypoint features as landmarks. Although impressive, MonoSLAM was limited to 100 features to maintain real-time operation given the computational budget at the time which limited the scale of the areas that could be mapped. The same sparse feature-based approach to image processing has since been adopted in many of the monocular SLAM successes, although some of the most common approaches have moved to bundle adjustment and graph SLAM approaches.

A notable later monocular SLAM advancement was Parallel Tracking and Mapping (PTAM) [Klein and Murray, 2007] so called because the tracking and mapping tasks were separated into two parallel threads. As tracking was relatively inexpensive it could be performed in real-time on each frame whereas the more expensive joint optimisation using bundle adjustment occurred at a slower frequency in a parallel background thread. The area that PTAM could operate in was also relatively limited. In recent years one of the most popular descendents of these monocular SLAM methods is called ORB SLAM [Mur-Artal and Tardós, 2014] which overcame some of the past limitations and is used in many applications due to its robustness and ease of use. Although this thesis focuses on dense semantic methods, the robustness of feature-based matching across wide viewpoint changes led to its use in the relocalisation approach described in Chapter 6.

Keypoint features themselves aim to be repeatedly detected and maintain stable descriptors in the presence of image transformations. Research on keypoint features has led to a proliferation of options which can be selected from based on their performance in the task domain and the available computational budget. Some notable examples include SIFT [Lowe, 1999], SURF [Bay et al., 2006], ORB [Rublee et al., 2011] (a combination of the FAST keypoint detector [Rosten and Drummond, 2006] and BRIEF descriptor [Calonder et al., 2010]), and BRISK [Leutenegger et al., 2011]. The impact of Deep Learning on every area of computer vision in recent years is evident even here with research being conducted into learned keypoint features such as the Learned Invariant Feature Transform (LIFT) [Yi et al., 2016].

1.1.2 Dense SLAM

Sparse feature-based methods are robust methods for localisation that are heavily used to the present day but the sparsity of the map means they cannot directly map free space or produce reconstructions which are useful for obstacle avoidance and path planning. Research continues on sparse methods but there has been a shift in more recent years to exploring dense methods which reconstruct the entirety of the scene geometry. Some early research sought a middle ground by augmenting feature-based monocular SLAM with dense structure but the sparsity of the points resulted in rough reconstructions [Lovegrove, 2011]. Research has since moved to fully dense approaches which form the SLAM backbone to much of the research in this thesis.

Unlike the previously described ‘indirect’ feature-based tracking approaches commonly used in sparse SLAM, ‘direct’ methods aim to minimise the error of directly measurable quantities [Irani and Anandan, 1999], such as the total distance between points in two misaligned point clouds. While most dense systems use direct methods and most sparse systems use indirect methods there are exceptions, such as the Direct Sparse Odometry (DSO) system produced by [Engel et al., 2017]. An early and illustrative example of a dense direct approach can be seen in the [Lucas and Kanade, 1981] method for image alignment in which gradient methods ‘slide’ images over each other to minimise the total intensity difference.

[Newcombe et al., 2011b]’s Dense Tracking and Mapping (DTAM) system takes an incoming RGB video feed and uses it to estimate dense textured depth maps of the scene using a regularised photometric cost volume. This volume forms a dense model of the scene which can be rendered and used in a dense tracking step to estimate the camera pose in an iterative [Lucas and Kanade, 1981] fashion. To allow this approach to operate in real-time the availability of high performance consumer GPUs was essential. Calculating per-pixel costs, rendering reference images, and performing Jacobian reductions are all operations which can readily take advantage of the massively parallel nature of GPUs and they feature heavily in most dense SLAM systems to the present day.

Interestingly, GPUs are not the only technology developed for video-games that has been usefully appropriated by SLAM and robotics researchers. Dense methods have also been fuelled by the rise of commodity RGB-D cameras beginning with the Microsoft Kinect [Microsoft Corp, 2010] for the Xbox 360 games console. The widespread availability of commodity depth cameras was very rapidly applied to dense SLAM systems, beginning with the impressive system of [Henry et al., 2010] which used Iterative Closest Point (ICP) alignment for tracking and a surfel map representation in a large-scale loop closure capable SLAM system. A surfel is a small surface patch represented by a disc and it is a representation that has continued to be popular in dense SLAM systems [Keller et al., 2013, Whelan et al., 2015b].

More recently the ElasticFusion [Whelan et al., 2015b] system, which also used a surfel map representation, attempted to tackle some of the inherent difficulties posed by loop closure in dense SLAM systems by using a deformation graph to deform the map using methods borrowed from animation skeletons in computer graphics. ElasticFusion forms the foundation for the SLAM assisted data collection system presented in Chapter 3 and the CNN-based semantic annotation system Semantic-Fusion presented in Chapter 4. The challenge loop closure poses to dense systems is one of the driving forces behind the 'sparse' object-level map representation chosen for Fusion++ in Chapter 6.

Soon after the release of the Kinect, the very influential KinectFusion [Newcombe et al., 2011a] system was also developed. It used an implicit volumetric surface representation for reconstruction in the form of a TSDF [Curless and Levoy, 1996]. Depth measurements are fused into the voxels of the volume using a weighted average scheme and ICP is used for dense tracking. The spatial extent of the volume was quite limited due to the cubic scaling memory requirements, but the fusion approach very successfully mitigated the inherently noisy depth measurements from the Kinect and produced accurate and smooth reconstructions. For this reason it became a favoured method of choice for 3D reconstruction systems. Subsequent pieces of work based on KinectFusion have used or registered multiple shifted subvolumes to produce high-quality reconstructions over larger areas [Zhou et al., 2013, Whelan et al., 2012] and other work has further engineered the system [Prisacariu et al., 2014] and reduced the memory requirements directly with schemes such as Voxel Hashing [Nießner et al., 2013]. Our object reconstruction methodology in Chapter 6 closely follows the KinectFusion approach.

The many years of research into dense reconstruction and SLAM have provided off-the-shelf tools that, although certainly not perfect, in some ways commoditise SLAM. They form the springboard for much of the research in this thesis, and in the earlier chapters relatively straightforward additions to pre-existing dense SLAM systems are made in order to move beyond geometry to include semantic information.

1.2 Semantic SLAM

The dense geometry of 3D scene reconstructions allow for robotic obstacle avoidance, path planning, and also enables occlusion aware augmented reality applications. The advance beyond geometry to semantically aware maps greatly increases the range of tasks which can be approached by robotic agents, and at the same time provides higher level concepts to a map that can assist in providing an intuitive user interface for human-map interaction; it is much easier to talk about 'the kitchen' than to specify a location in terms of cartesian coordinates. The inclusion of semantic and task-specific information in a map is described by [Cadena et al., 2016] as a requirement of the current *robust-perception age* of SLAM and a more thorough survey of semantic SLAM is presented by [Kostavelis and Gasteratos, 2015].

The useful properties of task specific semantics in maps have long been recognised [Chatila and Laumond, 1985, Kuipers and Byun, 1991] with topological maps with semantically relevant nodes. Many of the initial approaches manually encoded the required task specific information into the maps. For example in [Thrun et al., 1999] an interactive robotic tour-guide called 'Minerva' was installed in a Smithsonian museum and navigated between 23 different exhibits explaining them to visitors. Even in a simple system such as this, geometry alone is not sufficient and certain task specific information was required. The two research threads that have predominated semantic maps can be roughly divided into those which focus on semantic segmentation for scene annotation and those which use object detection for object-level maps (there are of course also methods which straddle this boundary). The initial work of this thesis, described in Chapter 4, followed the dense semantic annotation approach, but as the limitations of what could be achieved with such a representation became clear our research focus shifted to the new paradigm of object-level and instance mapping which is covered in Chapter 6.

1.2.1 Semantic Scene Annotation

The early 2000s saw some of the earliest work to automatically label dense 3D semantic maps using a laser range finder [Nüchter et al., 2003]. The labelling was not performed during the mapping itself, instead the complete indoor 3D scene was mapped and then planar regions were extracted from the point cloud using Random Sample Consensus (RANSAC). The planar regions were then labelled using a manually designed constraint network based on relative orientations and positions of the planes into the four categories of wall, floor, ceiling, and door. [Limketkai et al., 2005] used a Relational Markov network and learned feature weights from a small annotated dataset to produce a semantically annotated map of a corridor from 2D line segment primitives. Later work by [Mozos et al., 2007] produced a semantically annotated 2D occupancy map from laser scans using an Adaboost classifier.

In dense 3D indoor scene segmentation much of the work continued to focus on map post-processing of semantics, rather than including them in a live SLAM system. [Rusu et al., 2008] registered dense point clouds from a laser scan into a stitched scene and then used region segmentation to group planar areas into cuboid objects such as cupboards. Unassigned regions close to the planar surface were extracted and their projection onto the plane was assigned to either a 'handle' or 'knob' based on their elongation.

Early work by [Brostow et al., 2008] avoided the requirement of a laser scan by using Structure from Motion (SfM) from multiple RGB images to produce point clouds of scenes. These point clouds provided geometric cues that were projected into the image plane and used as features in a random forest for 2D semantic segmentation. They also showed that combining these geometric cues with 2D appearance-based cues improved the resulting segmentation. Subsequent work by [Bao et al., 2012] combined scene annotation and object detection for batch semantic SfM mapping. They used a parts-based object detector [Felzenszwalb et al., 2010] and graph cut region classifier [Ladicky et al., 2010] to jointly solve labelling and reconstruction of semantic maps separated into regions of annotated surface or objects. [Koppula et al., 2011] produced one of the earliest dense semantic annotation systems using commodity RGB-D cameras that annotated both structural and object elements of a scene. They form segments of the map into nodes of a graphical model and used hand-crafted geometric and visual features as edge potentials to infer the final semantic labelling.

Dense near real-time semantically annotated scenes using commodity sensors appeared with the work of [Stückler et al., 2015] and [Hermans et al., 2014]. They both obtain per-pixel label predictions for incoming frames using Random Decision Forests and fused predictions from different viewpoints in a classic Bayesian framework. [Stückler et al., 2015] used a Multi-Resolution Surfel Map SLAM system capable of operating at 12Hz while [Hermans et al., 2014] did not use a full SLAM system with explicit loop closure and instead used camera tracking with a run-time performance of 5Hz. [Cavallari and Di Stefano, 2016c] approached the problem of dense annotation using a KinectFusion map and a CNN to provide the semantic predictions for annotating a 3D map with each voxel storing a single category and a score. The multi-GPU version of their system operates at 17Hz and the single GPU version operates at 5Hz. These approaches bring us to the work described in Chapter 4 which combines ElasticFusion and CNN predictions alongside a Bayesian update scheme for real-time (25Hz) dense semantic mapping.

1.2.2 Object-Level Mapping

For object-detection based semantic SLAM research, some of the earliest work was by [Galindo et al., 2005]. They used a laser range finder alongside sonar sensors to build an occupancy map and link objects to a conceptual semantic hierarchy to infer the room category. The objects themselves were known beforehand and detected via simple colour and shape cues. The earliest example of vision-based semantic SLAM was by [Castle et al., 2007] who built a 3D map of planar objects during live operation of a SLAM system using only a monocular camera. The SLAM system was based on MonoSLAM, but on top of this planar objects were recognised from a pre-generated database of SIFT descriptors and added to the 3D map. Later the same year [Ranganathan and Dellaert, 2007] used learned models of known objects as an object-level approach to place recognition.

A subsequent monocular SLAM system by [Civera et al., 2011] also used a database of known objects which could be inserted into the map, but they removed the planar constraint and modelled objects using SURF descriptors with a known spatial arrangement. In a work that used RGB-D data but was not designed for live SLAM operation, [Lai et al., 2012] used a view-based Support Vector Machine (SVM) object detector to reconstruct and label objects of interest by integrating predictions into a voxel representation. [Kim et al., 2012] produced another system not designed for in-the-loop live use but which contained many of the elements of dense indoor object-level maps. Dense 3D models are first captured into a database and later, within a scan of a room, these object instances are recognised and their pose estimated in order to construct a map of the known objects.

The progenitor and inspiration for much of the work in this thesis was the SLAM++ system developed by [Salas-Moreno et al., 2013]. It uses a database of dense reconstructions of objects and real-time recognition using Point-Pair Features. After estimating the object's pose using a Hough forest, it includes the detected object reconstructions within the loop of a dense SLAM system. These objects provide constraints for an object-level pose graph and they are used for loop closure and relocalisation. This is closely related to our approach in Chapter 6. Subsequent work on object discovery by [Choudhary et al., 2014] also localised the camera in an online manner with discovered objects that are used as centroidal landmarks in a pose graph. An example of a more recent object-oriented mapping approach is the Meaningful Maps approach of [Sünderhauf et al., 2017] where instances are discovered using bounding box detections from a CNN and an unsupervised geometric segmentation algorithm. They use a separate RGB-D SLAM system, ORB-SLAM2 [Mur-Artal and Tardós, 2015], for localisation and to produce point clouds of the discovered instances.

This section has made apparent the wide variety of approaches available for semantic segmentation and object detection, ranging from manual heuristics to random forests. In recent years Deep Learning and CNNs have become the dominant approach to semantic visual processing. Their dominance results from the significant performance improvements they have been empirically shown to offer along with the exciting promise of a flexible and general method of automated machine learning. In this thesis Deep Learning has been selected as the method to provide semantic understanding, and in the next section we will provide a brief historical background to these methods as well as an introduction to the current wave of research following the 2012 ImageNet competition.

1.3 A Very Brief History of Deep Learning

Deep Learning is the currently favoured term describing an area of research with a long history that has previously been known as cybernetics and connectionism [Goodfellow et al., 2016]. The approach is biologically inspired making use of Artificial Neural Networks (ANNs) which have their basis in early models of biological neurons. A helpful analogy of the link between modern Deep Learning and biological neurons may be seen between birds and planes; they both have wings but few practical planes flap and birds don't use jet engines for thrust. In the same way, artificial neurons mimic some of the basic characteristics of neurons such as multiple 'dentrite' inputs with an 'axon' output, but are a far cry from the complex biological models of neurons produced over 50 years ago [Hodgkin and Huxley, 1952].

Modern ANNs are composed of neurons that are almost identical to one of the earliest[2] artificial neuronal models called the 'perceptron' [Rosenblatt, 1958]. The perceptron performs a weighted summation of raw inputs in order to produce an internal 'activation.' In the perceptron model this activation goes through the heaviside step function to produce an output but modern artificial neurons (shown in Figure 1.3) now use a variety of non-linear activation functions.

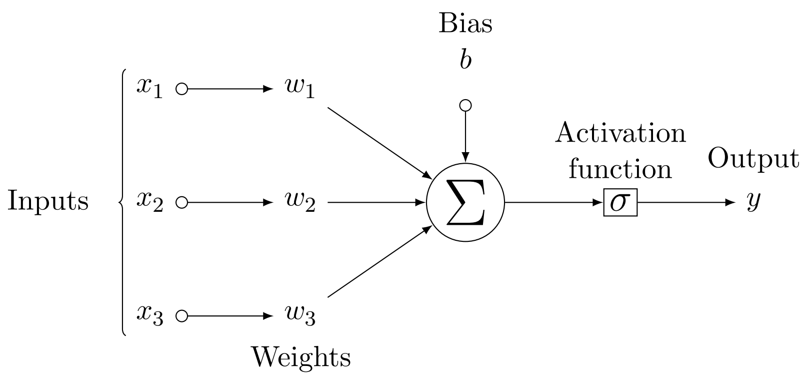


Figure 1.3: A diagram of a single artificial neuron. Modified from https://tex.stackexchange.com/questions/132444, original author https://tex.stackexchange.com/users/3954/gonzalo-medina. License: CC BY-SA 3.0.

The Multilayer Perceptron (MLP) is one of the simplest feed-forward deep learning network architectures. The term 'Multilayer Perceptron' is something of a misnomer; the network is composed of modern artificial neurons rather than being restricted

[2]The first artificial neuronal model was proposed by [McCulloch and Pitts, 1943]. It used binary activations but fixed identical weights and lacked a learning mechanism.

to perceptrons, and it consists of many neurons organised into layers rather than being a single 'multilayered' neuron. The neurons in each layer are connected to all of the neurons in the layer below but to none in its own layer. This configuration of connections is called 'fully-connected' and is shown in Figure 1.4.

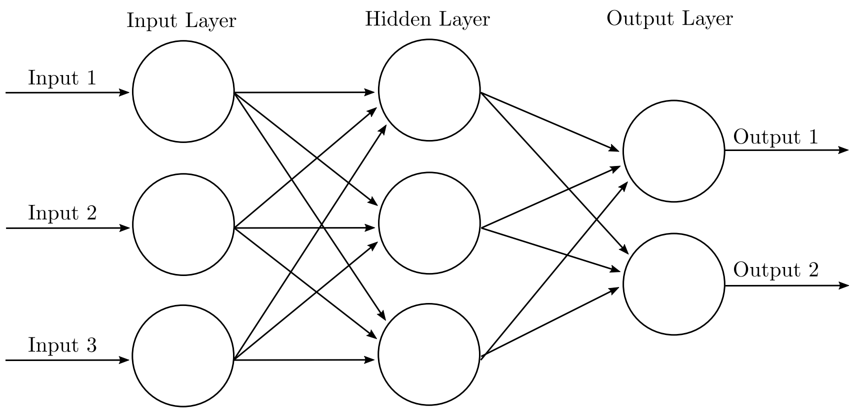


Figure 1.4: Diagram of a Multilayer Perceptron with three layers.

A specific type of Neural Network called a Convolutional Neural Network (CNN) has resulted in a dramatic leap forward in the performance of state-of-the-art algorithms in visual processing. Convolutional layers are also biologically inspired, this time from early research into the visual cortex of cats [Hubel and Wiesel, 1962]. This neurophysiology research provided the inspiration for the Neocognitron [Fukushima and Miyake, 1982] which has a similar tiled and layered architecture to modern CNNs but was not trained in a supervised manner using backpropagation, a technique which was only developed later [Rumelhart et al., 1986]. Convolutional layers stride the same learned image filter across an entire image to produce feature activations for the next layer; sharing the weights in this way both reduces the number of parameters and allows the same basic features such as edges and blobs to be shared across an image.

Modern Deep Learning has been described as 'representation learning' [LeCun et al., 2015]. Instead of hand-crafted feature extractors and careful engineering, the features in Deep Learning are learned directly from the raw data in a hierarchy of increasingly abstract layers. As will be discussed in Chapter 2 this hierarchy of composed features can be visualised in the layers of modern CNNs which first appeared with the LeNet-5 architecture of [LeCun et al., 1998] for document recognition. The current surge of interest in CNNs occurred much more recently, with their notable success in the ImageNet 2012 competition.

1.3.1 ImageNet 2012 Onwards

In the now seminal work by [Krizhevsky et al., 2012], a CNN was used to dramatically improve upon the state of the art in the 2012 ImageNet Large Scale Visual Recognition Competition (ILSVRC) [Russakovsky et al., 2015] with a top-5 error rate of 15% vs 26% for the next best entry. The CNN which became known as 'AlexNet' was remarkably similar to the LeNet-5 architecture. One of the key differences between these CNNs was scale; AlexNet had more convolutional layers (5 vs. 3) and parameters (60M vs 60k) and it was trained on 1.2M images.

The increase in scale of AlexNet was enabled by high performance General Purpose GPUs as well as the improved accessibility to GPUs via APIs such as CUDA (in fact the original implementation of AlexNet was in a framework called 'CUDA-ConvNet'[3]). This astounding empirical leap spurred others to adopt the same approach and since then the ILSVRC competition has been dominated by ever improving CNN entries [Szegedy et al., 2015, Simonyan and Zisserman, 2015, He et al., 2016].

CNNs' dominance has not been limited to image classification; it was also quickly applied to object detection. Some of the first examples such as Regions with CNN (R-CNN) [Girshick et al., 2014] simply leveraged image classification CNNs to classify and filter bounding boxes proposed by other methods. Soon after, CNNs were also used to propose the bounding boxes themselves in a system called 'Multibox' [Erhan et al., 2014]. Fully end-to-end learned object detectors then emerged which combined these approaches in Faster R-CNN [Ren et al., 2015]. Faster R-CNN was used as the basis for modern instance segmentation methods such as Mask R-CNN [He et al., 2017] (used in Chapter 6 of this thesis) which produces an additional binary instance mask output for each of the detected objects.

CNNs have also been applied to pixel-wise semantic image segmentation through a relatively simple modification to the classification network architecture [Long et al., 2015, Sermanet et al., 2014]. [Long et al., 2015] proposed converting the fixed size fully-connected layers at the top of a network into convolutional layers, with a classification kernel that could slide over the input. This process allows a 'heat-map' of per-pixel semantic predictions to be produced. They also proposed learned 'transpose convolutional' layers which allow spatially smaller feature maps to be

[3] Available at https://code.google.com/archive/p/cuda-convnet/.

upsampled to higher resolutions with learned filters. The coarse spatial information in the bottleneck motivated the use of 'skip' connections to supply fine-grained spatial information from lower layers to higher ones. CNNs of this variety will also be used frequently throughout this thesis.

In computer vision CNNs have been successfully applied to a plethora of problems such as optical flow estimation [Fischer et al., 2015], super-resolution [Kim et al., 2016], style transfer [Isola et al., 2017], and image restoration [Ulyanov et al., 2016a] to name but a few. The sphere of influence of CNNs extends beyond computer vision. At the beginning of this chapter we highlighted the achievement of AlphaGo [Silver et al., 2016], in which the vast search space of Go was reduced using predictions from trained CNNs with a 19×19 'image' based on the current board state. Perhaps of more real-world importance to future work on this thesis is the potential direct application of CNNs for learned robotic manipulation policies [Levine et al., 2016]. The applicability of Deep Learning methods to different problem domains and the flexibility with which different elements can be combined into optimisable structures is one of the reasons for the current excitement in the field, and its impact on the problem of Semantic SLAM is only just beginning.

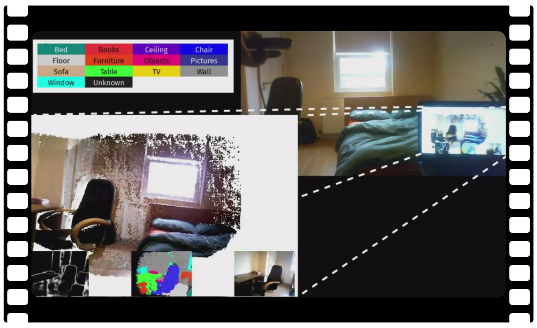
1.4 Publications

The work described in this thesis has resulted in the following publications:

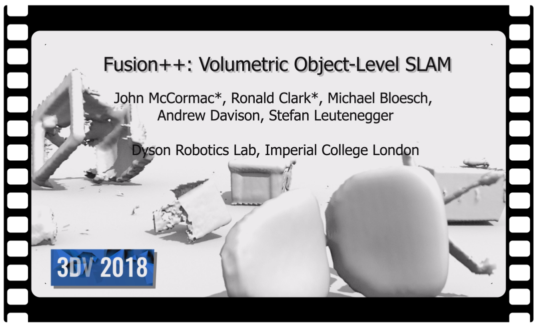
- SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks [McCormac et al., 2017a] (presented at ICRA 2017, patented).
- SceneNet RGB-D: Can 5M Synthetic Images Beat Generic ImageNet Pre-training on Indoor Segmentation? [McCormac et al., 2017b] (presented at ICCV 2017, featured in RSIP Computer Vision News[4]).
- Fusion++: Volumetric Object-Level SLAM [McCormac et al., 2018] (presented at 3DV 2018, patent pending).

[4]https://www.rsipvision.com/ICCV2017-Wednesday/https://www.rsipvision.com/ComputerVisionNews-2017November/#20

The following video material visualises and demonstrates the operation of some of the described methods:



SemanticFusion,
https://youtu.be/cGuoyNY54kU
ICRA 2017, *Singapore.*


Fusion++: Volumetric Object-Level SLAM
John McCormac*, Ronald Clark*, Michael Bloesch,
Andrew Davison, Stefan Leutenegger
Dyson Robotics Lab, Imperial College London
3DV 2018

Fusion++,
https://youtu.be/2luKNC03x4k
3DV 2018, *Verona, Italy.*

The following open-source code repositories and datasets have also been released for the research community:

- SemanticFusion: https://bitbucket.org/dysonroboticslab/semanticfusion
- SceneNet RGB-D:
  - The dataset: https://robotvault.bitbucket.io/scenenet-rgbd.html
  - Dataset tools: https://github.com/jmccormac/pySceneNetRGBD
  - Dataset generation: https://bitbucket.org/dysonroboticslab/scenenetrgb-d
  - CNN models: https://github.com/ankurhanda/pytorch-SceneNetRGBD

1.5 Thesis Structure

The rest of this thesis is structured as follows:

Chapter 2 introduces basic notation and the concepts used in dense SLAM and provides a primer on Convolutional Neural Networks (CNNs).

Chapter 3 describes the tool developed to annotate dense SLAM maps generated from RGB-D videos with semantic and instance labels to create training and validation datasets for video classification. It allows annotations in 3D to be reprojected to produce many annotated 2D frames.

Chapter 4 discusses SemanticFusion which produces a semantically labelled 3D map using a semantic segmentation probability map from a CNN and a dense surfel-based SLAM system called ElasticFusion. The performance of the fusion approach is validated on both the NYUv2 dataset and a small reconstruction dataset annotated with the software described in the previous chapter.

Chapter 5 describes the generation of an extremely large (5M image) synthetic dataset of video trajectories in indoor scenes. It provides pixel-accurate ground truth instance annotations, as well as the camera trajectory, depth map, and realistic ray-traced RGB images. We evaluate its potential to improve CNN performance by enabling domain specific pretraining.

Chapter 6 tackles the limitations found in previous chapters by using object detection and instance segmentation to build a map composed of discovered object instances. The objects are reconstructed in variable resolution TSDF volumes and are used as landmarks in a pose graph. Multiple 2D instance segmentations are fused into the volume to segment the objects.

Chapter 7 concludes with a discussion of the research performed so far and highlights areas for future research.

Chapter 2

Preliminaries

Contents

This section introduces the notational standard and preliminary background concepts that are used throughout this thesis.

2.1 Notation

Lower case bold letters represent N dimensional column vectors and upper case bold letters represent $N \times M$ dimensional matrices:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}, \qquad \mathbf{x}^\mathsf{T} = [x_1, x_2 \dots x_N], \qquad \mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1M} \\ x_{21} & x_{22} & \dots & x_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{NM} \end{bmatrix}. \tag{2.1}$$

In projective geometry it is useful to operate on vectors in *homogeneous coordinates*, where an additional final element is appended to the vector and operates to normalise the remaining elements, which are invariant when scaled by a common factor. A normalised homogeneous vector simply has an appended “1” element. Bold italic lower case is used to signify the homogeneous coordinates of a vector:

$$\boldsymbol{x} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}. \tag{2.2}$$

2.2 Transformations

In this work, rigid transformations between coordinate frames are used to describe the pose of the camera as well as objects in a scene. Coordinate frames are denoted by $\underrightarrow{\mathcal{F}}$ with an appropriate subscript describing the particular frame, e.g. the camera frame $\underrightarrow{\mathcal{F}}_C$ or world frame $\underrightarrow{\mathcal{F}}_W$. Points in 3D are described with reference to a particular coordinate frame; a point described in $\underrightarrow{\mathcal{F}}_C$ is denoted with a left-subscript ${}_C\mathbf{p}$.

Rigid transformations allow a point described in one coordinate frame to be described in a different coordinate frame. In 3D space a rigid body transformation has 6 degrees of freedom (DoF) consisting of a translation (3 DoF) and rotation (3 DoF). A common and useful, yet over-parametrised, representation for this transformation is a 4×4 homogeneous matrix,

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}, \tag{2.3}$$

consisting of a 3×3 rotation matrix, $\mathbf{R}$, and a translation vector $\mathbf{t} \in \mathbb{R}^3$. As rotation has 3 DoF, only a subset of the 9-element $\mathbf{R}$ produces a valid rotation. It must be within the Special Orthogonal group $\mathbf{R} \in \mathrm{SO}(3)$, such that $\mathbf{R}^\intercal\mathbf{R} = \mathbf{R}\mathbf{R}^\intercal = \mathbf{I}$ (where $\mathbf{I}$ is the identity matrix), with $\det(\mathbf{R}) = 1$. A 6 DoF transformation combining both $\mathbf{R} \in \mathrm{SO}(3)$ and $\mathbf{t} \in \mathbb{R}^3$ forms a member of the Special Euclidean group, SE(3). A group here is a mathematical group which comprises a set, G, such as the set of all SO(3) matrices, and an operator $\bullet$ which in the SO(3) case is matrix multiplication. Groups must also satisfy the four requirements of:

Closure:	if $a, b \in G$ then $a \bullet b \in G$.
Associativity:	if $a, b, c \in G$ then $(a \bullet b) \bullet c = a \bullet (b \bullet c)$.
Identity:	$I \in G$ exists such that for any $a \in G$, $a \bullet \mathbf{I} = \mathbf{I} \bullet a = a$.
Inverse:	$a^{-1} \in G$ exists for any $a \in G$ such that $a \bullet a^{-1} = a^{-1} \bullet a = \mathbf{I}$.

The major benefit of the homogeneous matrix form is that transformations can be applied with simple matrix multiplication. Here we denote a transformation matrix mapping a point ${}_A\mathbf{p}$ to ${}_B\mathbf{p}$ as $\mathbf{T}_{BA}$, such that,

$$ {}_B\boldsymbol{p} = \mathbf{T}_{BA}\ {}_A\boldsymbol{p}. \tag{2.4}$$

Note that in this case the $\mathbb{R}^4$ homogeneous coordinates of the points are used, as denoted by italics, as is required to multiply by a 4×4 transformation matrix. However this homogeneous form is often assumed for notational convenience in which case the italics are omitted:

$$ {}_B\mathbf{p} = \mathbf{T}_{BA}\ {}_A\mathbf{p}. \tag{2.5}$$

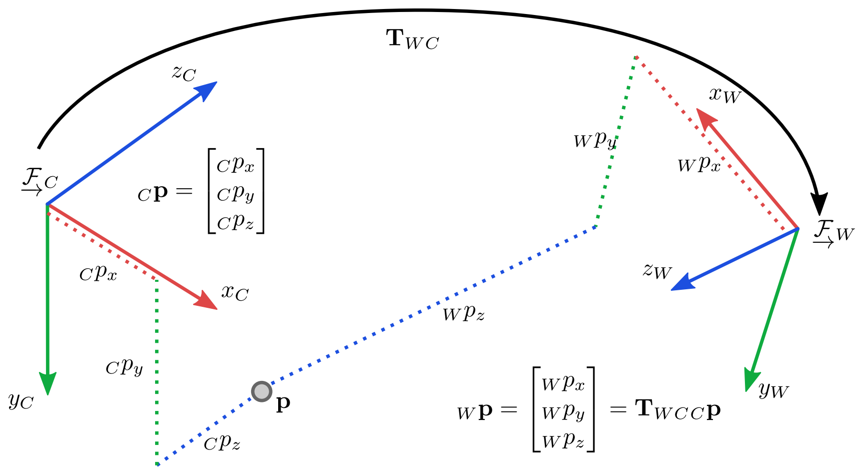
A visualisation of a point expressed in two coordinate frames, and with the transformation between them is shown in Figure 2.1. Transformations themselves can also be easily composed and are invertible:

$$\mathbf{T}_{CA} = \mathbf{T}_{CB}\mathbf{T}_{BA}, \qquad \mathbf{T}_{AB} = \mathbf{T}_{BA}^{-1} = \begin{bmatrix} \mathbf{R}_{BA}^\intercal & -\mathbf{R}_{BA}^\intercal\mathbf{t}_{BA} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{2.6}$$

In this work the standard format for camera poses will be the transformation from camera to world $\mathbf{T}_{WC}$ (and similarly for objects $\mathbf{T}_{WO}$). In SLAM for convenience $\underrightarrow{\mathcal{F}}_W$ is often defined to coincide with the pose of the camera in the first frame of a sequence, however any arbitrary frame of reference can be chosen.

2.2.1 Minimal Parametrisations and Lie Algebra

There exist numerous parametrisations of transformations which require fewer parameters than the homogeneous transform. These forms are also used within this



$\mathbf{T}_{WC}$
z_C
$\underrightarrow{\mathcal{F}}_C$
${}_C\mathbf{p} = \begin{bmatrix} {}_Cp_x \\ {}_Cp_y \\ {}_Cp_z \end{bmatrix}$
${}_Cp_x$
x_C
${}_Cp_y$
y_C
$\mathbf{p}$
${}_Cp_z$
${}_Wp_z$
${}_Wp_y$
x_W
${}_Wp_x$
$\underrightarrow{\mathcal{F}}_W$
z_W
y_W
${}_W\mathbf{p} = \begin{bmatrix} {}_Wp_x \\ {}_Wp_y \\ {}_Wp_z \end{bmatrix} = \mathbf{T}_{WC}\,{}_C\mathbf{p}$

Figure 2.1: A depiction of a point $\mathbf{p}$ expressed in two coordinate frames, $\underrightarrow{\mathcal{F}}_W$ and $\underrightarrow{\mathcal{F}}_C$, with the transformation between the two, $\mathbf{T}_{CW}$.

thesis, most notably when pose optimisation is required. As discussed above, the homogeneous form for rotations is highly over-parametrised with 9 parameters and 3 DoF. Only a small subspace within the 9-dimensional parameter-space produces a valid rotation matrix. In optimisation, when rotations are iteratively updated, it is simpler to optimise in a space that naturally constrains the solution to always produce a valid rotation, rather than attempting the more complicated approach of adhering to or verifying multiple non-linear constraints to keep the matrix within the SO(3) group.

Some examples of minimal parametrisations of 3D rotations are Euler angles, angle-axis, and quaternions. Each of these parameterisations can be useful and particularly suited to a given task. As our aim is to perform optimisation within the SO(3) group we require derivatives with respect to the transformation parameters in order to use gradient methods. It is therefore useful to understand that the SO(3) group is in fact a Lie group with an accompanying Lie algebra which is the local derivative of the SO(3) group manifold. A number of useful resources are available which provide more detailed motivation and mathematical underpinnings to the practical use of Lie groups in computer vision [Eade, 2014, Drummond, 2014, Bloesch et al., 2016]. Here we provide just a brief high-level summary for completeness.

A Lie group is a mathematical group, as described above, that also forms a smooth differentiable manifold. As a smooth manifold, the Lie group has an associated Lie algebra, $\mathfrak{so}(3) \in \mathbb{R}^3$, which is the local tangent space around the identity of the group. The Lie algebra of the group is a particularly natural space in which to perform iterative updates on the group element. It forms a vector space with the same dimensionality as the number of DoF of the group, $\mathfrak{so}(3) \in \mathbb{R}^3$, and so each step can freely move within the entire vector space. There also exists an exponential map that allows any point in the vector space, here denoted $\boldsymbol{\zeta} \in \mathbb{R}^3$, to be mapped back exactly into a group transform, $\exp : \mathfrak{g} \rightarrow G$. For the SO(3) group the exponential map has a closed form called the Rodriguez formula which can in practice be efficiently calculated. The exponential map can also be inverted using the logarithm to map a group member to the tangent vector space, $\log : G \rightarrow \mathfrak{g}$.

A final important property is that a local tangent space exists around every group member and it is possible to transform a vector in the tangent space (such as that around the identity) to the tangent space around any other member, $\mathbf{R}$, through the *Adjoint* operator:

$$\exp(\mathrm{Adj}_{\mathbf{R}}\boldsymbol{\zeta}) = \mathbf{R}\exp(\boldsymbol{\zeta})\mathbf{R}^{-1}. \tag{2.7}$$

A nice property of the Adjoint transformation is that it is linear. In the SO(3) case the Adjoint is the same as the applied rotation matrix $\mathrm{Adj}_{\mathbf{R}} = \mathbf{R}$. In Chapter 6 this identity will be of use when transforming an ICP system defined with left-perturbations as shown in Equation 2.8 to a pose graph system which uses right-perturbations [Kümmerle et al., 2011].

To apply local perturbations to a member of the group $\mathbf{T}$, the $\boxplus$ (boxplus) operator is used. It is a generalisation of vector addition to non-Euclidean spaces:

$$\mathbf{T} \boxplus \boldsymbol{\zeta} \triangleq \exp(\boldsymbol{\zeta})\mathbf{T}. \tag{2.8}$$

In the case of the translation group T(3) which is already a simple vector space this operator simplifies to standard vector addition. In more complex spaces such as for the SO(3) group and its Lie algebra this operator requires exponentiation to a 3×3 rotation matrix using the Rodriguez formula followed by matrix multiplication. Analogously to the addition operator, generalised subtraction is performed with the $\boxminus$ (boxminus) operator such that:

$$\mathbf{T}_1 \boxminus \mathbf{T}_2 \triangleq \log(\mathbf{T}_1\mathbf{T}_2^{-1}). \tag{2.9}$$

These operations work together as one may intuitively expect:

$$(\mathbf{T} \boxplus \mathbf{0}) = \mathbf{T}, \tag{2.10}$$

$$(\mathbf{T} \boxplus \boldsymbol{\zeta}) \boxminus \mathbf{T} = \boldsymbol{\zeta}, \tag{2.11}$$

$$\mathbf{T}_1 \boxplus (\mathbf{T}_2 \boxminus \mathbf{T}_1) = \mathbf{T}_2. \tag{2.12}$$

For optimisation of general rigid body transformations the SE(3) group is used. The derivative of a transformed point with respect to the Lie algebra parameters which transform it is of particular importance for SLAM and dense tracking:

$${}_B\mathbf{p} = (\mathbf{T}_{BA} \boxplus \boldsymbol{\zeta})_A\mathbf{p}. \tag{2.13}$$

The 3×6 Jacobian of ${}_B\mathbf{p}$ with respect to $\boldsymbol{\zeta}$ is:

$$\frac{\partial_B\mathbf{p}}{\partial\boldsymbol{\zeta}}|_{\boldsymbol{\zeta}=0} = (\mathbf{I}| -_B \mathbf{p}_\times), \tag{2.14}$$

where $\mathbf{p}_\times$ is the skew-symmetric cross product matrix of $\mathbf{p}$:

$$\mathbf{p}_\times = \begin{bmatrix} 0 & -p_3 & p_2 \\ p_3 & 0 & -p_1 \\ -p_2 & p_1 & 0 \end{bmatrix}. \tag{2.15}$$

2.3 Cameras

The systems developed in this thesis are based on commodity RGB-D sensors such as the Microsoft Kinect and Asus Xtion Pro Live. These sensors can provide a 30Hz VGA (640×480) video stream of dense depth readings which are registered on-board the device to the associated RGB frames. The underlying technology of both of these sensors was developed by PrimeSense [Garcia and Zalevsky, 2007]. It relies on an infrared speckle pattern projected from the device and captured by an Infrared sensor offset along the sensor x-axis. The captured pattern is compared against reference patterns from surfaces captured at known distances to compute the depth. Figure 2.2 shows the Asus Xtion Pro Live camera in an example scene. The captured RGB image and Infrared image of the projected speckle pattern can be seen alongside the final depth map.

2.3.1 Pinhole Camera Model

The standard pinhole camera model (Figure 2.3) is used to model this RGB-D camera in the SLAM systems developed in this thesis. It is also used when generating

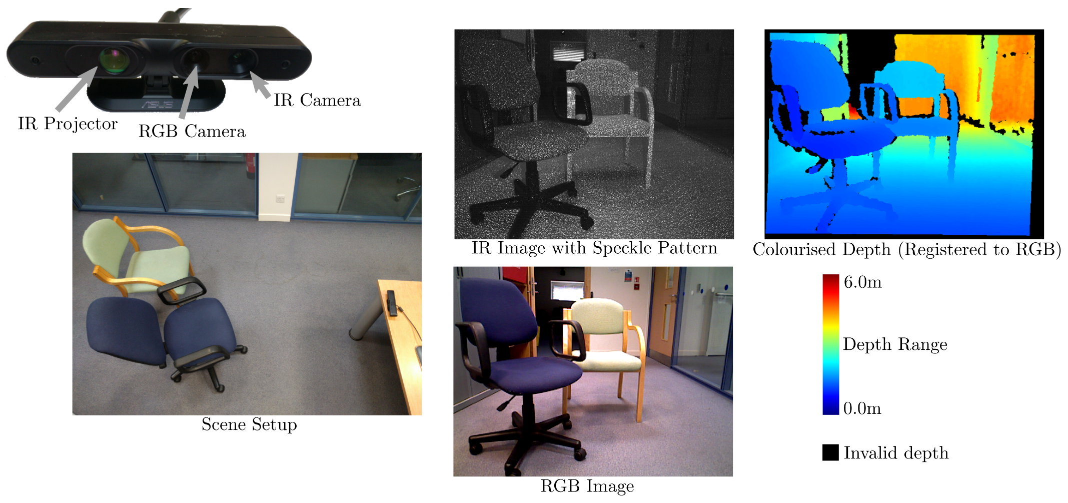


Figure 2.2: The Asus Xtion Pro Live capturing a scene in RGB-D.

the synthetically rendered dataset described in Chapter 5. The pinhole camera model describes the projection of 3D points, ${}_C\mathbf{p}$, into 2D pixel coordinates denoted as $\mathbf{u}$. The perspective projection function, denoted $\boldsymbol{\pi}$, projects a 3D point in camera coordinates ${}_C\mathbf{p} = [x, y, z]^\mathsf{T}$ to homogeneous coordinates on the unit plane ${}_C\mathbf{p}' = [x', y', 1]^\mathsf{T}$:

$$ {}_C\mathbf{p}' = \boldsymbol{\pi}\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}. \tag{2.16} $$

These coordinates are then mapped to pixel coordinates using the 3×3 intrinsic matrix denoted $\mathbf{K}$, which in this case is of the form:

$$ \mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.17} $$

The focal lengths f_x and f_y are measured in pixels and can therefore be different for each axis as the physical width and height of a pixel are not always equal. The offsets c_x and c_y account for the top left corner of the image being (0,0) in pixel coordinates, rather than having an origin at the principal point. Combining these operations the homogeneous pixel coordinates of a projected point becomes,

$$ \boldsymbol{u} = \mathbf{K}\boldsymbol{\pi}({}_C\mathbf{p}) = \begin{bmatrix} \frac{f_x x}{z} + c_x \\ \frac{f_y y}{z} + c_y \\ 1 \end{bmatrix}. \tag{2.18} $$

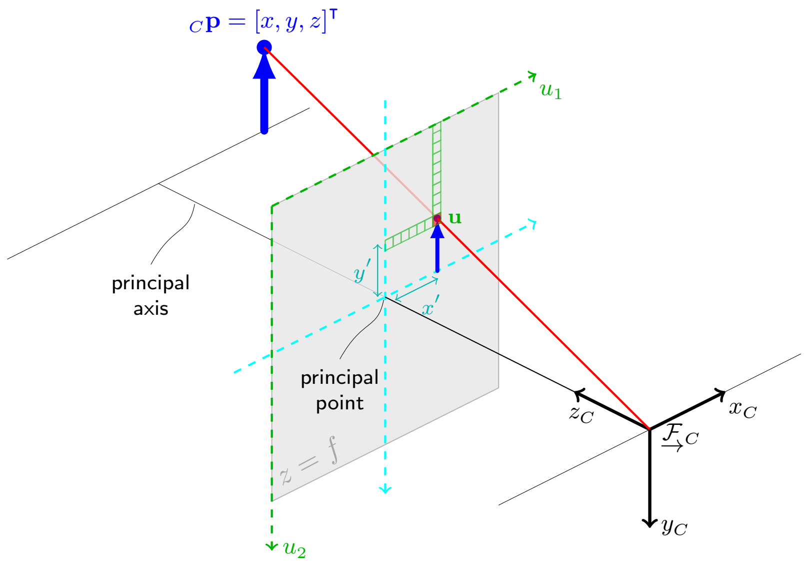


Figure 2.3: An illustration of the geometry of perspective projection. Modified from https://tex.stackexchange.com/questions/96074, original author https://tex.stackexchange.com/users/22653/perr0. License: CC BY-SA 3.0.

Upper case letters denote image intensity as a function of pixel coordinates, such that $I(\mathbf{u})$ refers to the intensity of image I at pixel coordinates $\mathbf{u}$. As $\mathbf{u}$ is continuous but I is discrete, bilinear interpolation or nearest neighbour sampling may be used to evaluate the final intensity value of the function.

2.3.2 Back-projection

Perspective projection as defined in Eq. 2.18 is not an invertible function as the depth dimension has been collapsed rendering all of the points along a ray path indistinguishable. However, given a dense depth map from an RGB-D sensor, denoted D, and assuming a pinhole camera model, we can recover the collapsed z-dimension to invert the projection equation and estimate the original 3D point in $\underrightarrow{\mathcal{F}}_C$:

$$ {}_C\mathbf{p} = \boldsymbol{\pi}^{-1}(\mathbf{K}^{-1}\boldsymbol{u}, D(\mathbf{u})), \tag{2.19}$$

where,

$$\mathbf{K}^{-1} = \begin{bmatrix} \frac{1}{f_x} & 0 & -\frac{c_x}{f_x} \\ 0 & \frac{1}{f_y} & -\frac{c_y}{f_y} \\ 0 & 0 & 1 \end{bmatrix}, \qquad \boldsymbol{\pi}^{-1}(\boldsymbol{u}, D(\mathbf{u})) = \begin{bmatrix} D(\mathbf{u})\boldsymbol{u}_1 \\ D(\mathbf{u})\boldsymbol{u}_2 \\ D(\mathbf{u}) \end{bmatrix}. \tag{2.20}$$

This simplifies to,

$$ {}_{C}\mathbf{p} = \begin{bmatrix} D(\mathbf{u})\frac{\boldsymbol{u}_1 - c_x}{f_x} \\ D(\mathbf{u})\frac{\boldsymbol{u}_2 - c_y}{f_y} \\ D(\mathbf{u}) \end{bmatrix}. \tag{2.21} $$

This operation can be performed for each pixel within a valid depth image to produce a point cloud or 'vertex map', as visualised in Figure 2.4. The depth map is registered to the RGB image on the device, and the association of pixel locations is used to colour the points in the point cloud. We use the default camera intrinsics provided by the public ElasticFusion implementation in the experiments presented in Chapters 3 & 4. In the evaluation presented in Chapter 6 we use the ROS default camera intrinsic parameters as recommended in the RGB-D SLAM Benchmark documentation [Sturm et al., 2012].[1]

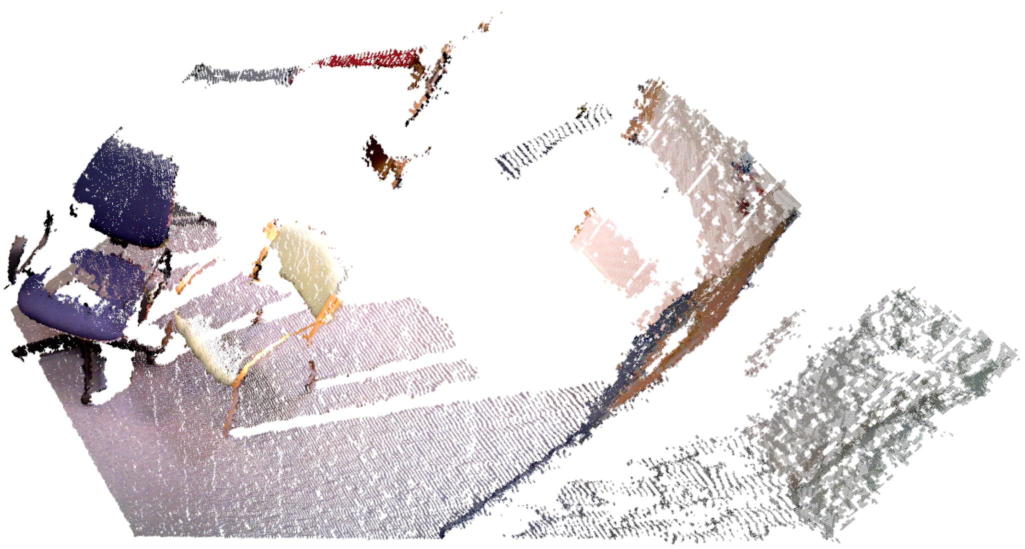


Figure 2.4: The back-projected point cloud from the scene in Figure 2.2.

2.4 Non-Linear Least Squares Optimisation

Non-Linear Least Squares (NLLS) optimisation is frequently used in SLAM to estimate the pose of a camera from a series of measurements and also to optimise the map elements. For a given set of state parameters, $\mathbf{x} \in \mathbb{R}^p$, such as the state of the current camera pose or map elements, the aim of NLLS is to find the parameters, $\hat{\mathbf{x}}$, which minimise the squared residuals $\mathbf{r}(\mathbf{z}, \mathbf{x})$ between the model's N predicted

[1] For details see https://vision.in.tum.de/data/datasets/rgbd-dataset/file_formats.

observations, $\mathbf{h}(\mathbf{x})$, and the actual measurements observed, denoted by $\mathbf{z}$:

$$E(\mathbf{x}) = \mathbf{r}(\mathbf{z},\mathbf{x})^\mathsf{T}\mathbf{W}\mathbf{r}(\mathbf{z},\mathbf{x}), \tag{2.22}$$

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}}\, E(\mathbf{x}), \tag{2.23}$$

where $\mathbf{W}$ is a symmetric weighting matrix specifying the relative importance of different residuals which can be left as the identity $\mathbf{I}$ in the absence of explicit weights. The residual function can be calculated via simple subtraction:

$$\mathbf{r}(\mathbf{z},\mathbf{x}) = \mathbf{z} - \mathbf{h}(\mathbf{x}), \tag{2.24}$$

or through a more advanced error term such as the point-to-plane error function described below in Section 2.4.1.

If Equation 2.24 is a linear function of $\mathbf{x}$ such as $\mathbf{r}(\mathbf{z},\mathbf{x}) = \mathbf{z} - \mathbf{A}\mathbf{x}$ then a closed-form solution of $\hat{\mathbf{x}}$ can be calculated using the *normal equation*. This can be derived by finding the point where $\frac{\partial E}{\partial \mathbf{x}} = 0$:

$$\hat{\mathbf{x}} = (\mathbf{A}^\mathsf{T}\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^\mathsf{T}\mathbf{W}\mathbf{z}. \tag{2.25}$$

Unfortunately for most SLAM problems the measurement function is non-linear and so an iterative optimisation scheme such as Gauss-Newton or Levenberg-Marquardt must be employed to solve the least squares problem. The convergence of these approaches is not guaranteed and it may converge to a local minimum [Eade, 2009].

The Gauss-Newton system begins with an initial guess of the parameters, $\mathbf{x}_0$. It then operates by iteratively linearising $\mathbf{r}(\mathbf{z},\mathbf{x})$ about the current guess using a first-order Taylor approximation and the $N \times p$ Jacobian $\mathbf{J} = \frac{\partial \mathbf{r}(\mathbf{z},\mathbf{x})}{\partial \mathbf{x}}$:

$$\mathbf{r}(\mathbf{z},\mathbf{x} \boxplus \boldsymbol{\delta}) \approx \mathbf{r}(\mathbf{z},\mathbf{x}_0) + \mathbf{J}\boldsymbol{\delta}. \tag{2.26}$$

Substituting this linearised form into the error function given in Equation 2.22 and setting the derivative to equal zero allows us to solve the system in the same manner as for linear least squares using the normal equation:

$$E(\mathbf{x} \boxplus \boldsymbol{\delta}) = (\mathbf{r} + \mathbf{J}\boldsymbol{\delta})^\mathsf{T}\mathbf{W}(\mathbf{r} + \mathbf{J}\boldsymbol{\delta}), \tag{2.27}$$

$$\frac{\partial E}{\partial \boldsymbol{\delta}} = 2(\mathbf{r} + \mathbf{J}\boldsymbol{\delta})^\mathsf{T}\mathbf{W}\mathbf{J}, \tag{2.28}$$

$$\mathbf{0} = \mathbf{r}^\mathsf{T}\mathbf{W}\mathbf{J} + \boldsymbol{\delta}^\mathsf{T}\mathbf{J}^\mathsf{T}\mathbf{W}\mathbf{J}, \tag{2.29}$$

$$\boldsymbol{\delta} = -(\mathbf{J}^\mathsf{T}\mathbf{W}\mathbf{J})^{-1}\mathbf{J}^\mathsf{T}\mathbf{W}\mathbf{r}. \tag{2.30}$$

After calculating the update it is applied to the state estimate $\mathbf{x}_1 = \mathbf{x}_0 \boxplus \boldsymbol{\delta}$ and the process is iterated usually until the updates becomes sufficiently small, or a certain number of iterations is reached. Here $(\mathbf{J}^\intercal \mathbf{W} \mathbf{J})$ is an approximation to the Hessian of Equation 2.22 with respect to the parameters $\mathbf{x}$. It is an approximation which only relies on the Jacobian of $\mathbf{r}(\mathbf{z}, \mathbf{x})$ as higher-order terms have been dropped by the first-order Taylor approximation in Equation 2.26.

The use of least squares can also be motivated by a probabilistic formulation of the problem. Under the common assumption of K independent measurements with additive zero-mean normally distributed noise:

$$\mathbf{z}_k = \mathbf{h}_k(\mathbf{x}) + \epsilon_k, \tag{2.31}$$

where $\epsilon_k \sim \mathcal{N}(0, \boldsymbol{\Sigma}_k)$. The maximisation of the likelihood function $\mathcal{L}(\mathbf{x}) = p(\mathbf{z}|\mathbf{x})$ (which will also be Gaussian) can be formulated as a least squares problem:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmax}} \, \mathcal{L}(\mathbf{x}), \tag{2.32}$$

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmax}} \prod_{k=1}^{K} p(\mathbf{z}_k|\mathbf{x}), \tag{2.33}$$

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmax}} \prod_{k=1}^{K} \exp(-(\mathbf{z}_k - \mathbf{h}_k(\mathbf{x}))^\intercal \boldsymbol{\Sigma}_k^{-1} (\mathbf{z}_k - \mathbf{h}_k(\mathbf{x}))). \tag{2.34}$$

As the logarithm is monotonic we can use it to change the multiplicative terms into additive ones without changing the argmax. Finally converting from an $\operatorname{argmax}_{\mathbf{x}}$ to an $\operatorname{argmin}_{\mathbf{x}}$ by negating the equation forms the system into a weighted least squares problem. We can also see that in this setting the $\mathbf{W}$ matrix above is specified by the inverse covariance matrix of the likelihood function, $\boldsymbol{\Sigma}^{-1}$:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \sum_{k=1}^{K} (\mathbf{z}_k - \mathbf{h}_k(\mathbf{x}))^\intercal \boldsymbol{\Sigma}_k^{-1} (\mathbf{z}_k - \mathbf{h}_k(\mathbf{x})). \tag{2.35}$$

In practice outlier measurements that are not distributed according to the assumed Gaussian noise model occur. The squared residual of these outliers can have particularly large and detrimental effects on the optimised parameters. Therefore some form of additional robustness measures are often also employed, such as outlier rejection or robust loss functions like the Huber loss, which mitigate large squared error terms. In other cases more extreme forms of robust regression can be used instead of least squares to fit model parameters, such as Random Sample Consensus (RANSAC). These methods will be highlighted when they are used in later chapters.

2.4.1 Iterative Closest Point

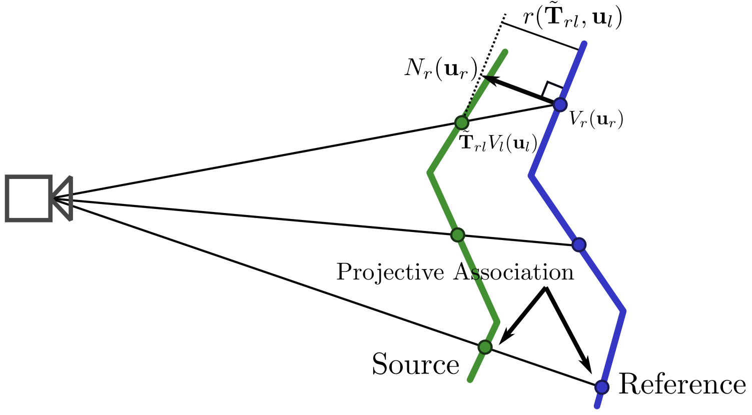
A common approach in real-time dense SLAM systems is to alternate between tracking and mapping. Assuming a reconstructed dense reference map r, Iterative Closest Point (ICP) [Besl and McKay, 1992] is a NLLS method which can be used to register it to an incoming live frame l by estimating the transform between the two [Newcombe et al., 2011b]. In this case that transform describes the estimated camera pose, $\tilde{\mathbf{T}}_{C_r C_l}$.

As a brief overview, the classic ICP algorithm begins by associating the closest points in the source point cloud V_l (in our case constructed from a live frame in $\underrightarrow{\mathcal{F}}_{C_l}$) and the reference (map) point cloud V_r, here with vertices defined in $\underrightarrow{\mathcal{F}}_W$. NLLS is used to find the rigid body transform which minimizes the sum of the squared distances between these corresponding points. After optimisation the estimated transform is applied to the source point cloud, a new set of associations are calculated based on the closest points and the procedure is repeated.

The ICP algorithm explained here is modified from the above in a number of ways as depicted in Figure 2.5 in a 2D cross-section. The modified version is called Fast ICP [Rusinkiewicz and Levoy, 2001]. For performance, projective data association [Blais and Levine, 1995] is used to calculate dense correspondences between the two images rather than searching for the closest two points in terms of Euclidean distance. Each pixel in the live depth image is first back-projected to a 3D point in the live camera frame, $\underrightarrow{\mathcal{F}}_{C_l}$, using Equation 2.21. This produces a vertex map, $V_l(\mathbf{u}_l)$, of 3D points for each pixel location, $\mathbf{u}_l$. The current estimate of the camera pose, $\tilde{\mathbf{T}}_{WC_l}$, is then used to transform these points into the reference frame, $\underrightarrow{\mathcal{F}}_{C_r}$. Correspondences to reference image pixels, $\boldsymbol{u}_r$, can then be calculated by projecting the live points into the reference image plane:

$$\boldsymbol{u}_r = \mathbf{K}\boldsymbol{\pi}(\mathbf{T}_{WC_r}^{-1}\tilde{\mathbf{T}}_{WC_l}V_l(\mathbf{u}_l)). \tag{2.36}$$

The error metric is also modified to be the point-to-plane error [Chen and Medioni, 1992] using the reference surface normal. It has been found to work well in practice with projective data association, as it allows the surfaces to 'slide' over each other. The surface normal may be estimated from an implicit surface representation such as a TSDF or can be calculated from a dense depth image by using the cross-product



$r(\tilde{\mathbf{T}}_{rl}, \mathbf{u}_l)$
$N_r(\mathbf{u}_r)$
$V_r(\mathbf{u}_r)$
$\tilde{\mathbf{T}}_{rl}V_l(\mathbf{u}_l)$
Projective Association
Source
Reference

Figure 2.5: A visualisation of projective data association and the point-plane residual error used in Fast ICP [Rusinkiewicz and Levoy, 2001].

of neighbouring pixels' 3D points:

$$N_r(\mathbf{u}) = (V_r([u_1+1\ u_2]) - V_r([u_1\ u_2])) \times (V_r([u_1\ u_2+1]) - V_r([u_1\ u_2])). \quad (2.37)$$

The residual is calculated by projecting the distance between two corresponding vertices onto the surface normal vector of the reference image:

$$r_{\text{icp}}(\tilde{\mathbf{T}}_{WC_l}, \mathbf{u}_l) = N_r(\mathbf{u}_r) \cdot (V_r(\mathbf{u}_r) - \tilde{\mathbf{T}}_{WC_l} V_l(\mathbf{u}_l)). \quad (2.38)$$

As already mentioned NLLS methods are quite sensitive to outliers and so only a subset $V_{\text{valid}} \subseteq V_l$ of valid vertices are used in the final error function. Filtering criteria may place hard limits on the difference in distances $\|V_r(\mathbf{u}_r) - \tilde{\mathbf{T}}_{WC_l} V_l(\mathbf{u}_l)\|_2$ or the normal divergence $N_r(\mathbf{u}_r) \cdot N_l(\mathbf{u}_l)$ between two corresponding points. The final ICP error function becomes:

$$E_{\text{icp}}(\tilde{\mathbf{T}}_{WC_l}) = \sum_{\mathbf{u}_l \in V_{\text{valid}}} r_{\text{icp}}(\tilde{\mathbf{T}}_{WC_l}, \mathbf{u}_l)^2. \quad (2.39)$$

This NLLS problem can then be solved using the Gauss-Newton algorithm. Using t to denote the Gauss-Newton iteration index we linearise about the previous estimate, $\bar{\mathbf{T}}^t_{WC_l}$, and optimise for the update perturbation, $\boldsymbol{\zeta}^t$, where $\mathbf{T}^{t+1}_{WC} = \bar{\mathbf{T}}^t_{WC} \boxplus \boldsymbol{\zeta}^t$. Each row of the $|V_{\text{valid}}| \times 6$ Jacobian, $\mathbf{J}_{\text{icp}}$, corresponds to the residual of a given $\mathbf{u}_l \in V_{\text{valid}}$. It can be calculated from Equations 2.14 & 2.38 using the chain rule:

$$\mathbf{J}_{\text{icp}} = \frac{\partial r_{\text{icp}}(\boldsymbol{\zeta}, \mathbf{u}_l)}{\partial \boldsymbol{\zeta}}|_{\boldsymbol{\zeta}=0} = -[N_r^{\intercal}(\mathbf{u}_r), (V_l(\mathbf{u}_l) \times N_r(\mathbf{u}_r))^{\intercal}], \quad (2.40)$$

with iterative normal equation updates of:

$$\boldsymbol{\zeta}^t = -(\mathbf{J}_{\text{icp}}^\intercal \mathbf{J}_{\text{icp}})^{-1} \mathbf{J}_{\text{icp}}^\intercal \mathbf{r}_{\text{icp}}. \tag{2.41}$$

The number of measurements in dense ICP is approximately the number of pixels in the image. However while the Jacobian is large ($|V_{\text{valid}}| \times 6$), the final system of equations which must be solved is only 6×6 as shown in Equation 2.41. Section 2.6 of this chapter will introduce GPU programming which provides the capability to efficiently perform the calculations to reduce the error functions Hessian approximation $\mathbf{J}_{\text{icp}}^\intercal \mathbf{J}_{\text{icp}}$ and the Jacobian $\mathbf{J}_{\text{icp}}^\intercal \mathbf{r}_{\text{icp}}$ into sizes 6×6 and 6 respectively. It is then possible to efficiently copy this small amount of data to the CPU where it can be solved using a variety of approaches such as Cholesky Decomposition (CD) or Singular Value Decomposition (SVD) which are well supported in libraries such as Eigen [Guennebaud et al., 2010]. After solving for the update it is applied to the current estimate:

$$\tilde{\mathbf{T}}_{WC_l}^{t+1} = \tilde{\mathbf{T}}_{WC_l}^{t} \boxplus \boldsymbol{\zeta}^t. \tag{2.42}$$

Instead of simply repeating this process at the same resolution, a coarse-to-fine approach is generally adopted [Bergen et al., 1992]. This improves the speed of convergence and also makes the optimisation itself more robust by providing a wider basin of convergence. As a specific example, in a three-level coarse-to-fine pyramid, a 640×480 reference and source image may be half-sampled to 320×240 and 160×120. ICP is first performed on the coarsest 160×120 resolution to produce an initial estimate the transform which is used as the initial guess for the finer 320×240 resolution.

2.5 Convolutional Neural Networks

Dense SLAM provides the first key component for automatic 3D scene understanding, namely a map reconstruction and camera localisation. The second required component is automated semantic understanding of the reconstructed map itself. In this thesis the semantic understanding component is provided by recent advancements in Deep Learning and Convolutional Neural Networks (CNNs). As discussed in Chapter 1 CNNs are a biologically inspired hierarchical approach to visual processing that have become the dominant method in Computer Vision for Image Classification, Semantic Segmentation, and Object Detection. Although only a subset of the wider area of study of Deep Learning and Artificial Neural Networks, a huge range of CNNs have been developed, each with different architectures and custom layers all of which can be composed in myriad ways.

In its simplest form a feedforward CNN is composed of sequential layers of convolutional filters followed by non-linearities. Figure 2.6 illustrates this layered structure of CNNs and groups convolutions together with the subsequent non-linearity (here a ReLU which is described later in this chapter). As will shortly be discussed, convolutions are used to process smaller receptive field 'patches' of an image with the same filter in a tiled fashion. In this way the parameters of the filter are 'shared' across the whole image. This is a useful trait because it not only reduces the number of parameters thus reducing overfitting but it also naturally provides translational 'equivariance' between the input image and the output feature map; if the image is shifted to the left the same features will be computed but they will equally be shifted to the left. The final result in classification tasks will also hopefully be translation invariant by producing an identical (non-shifted) classification for the translated input.

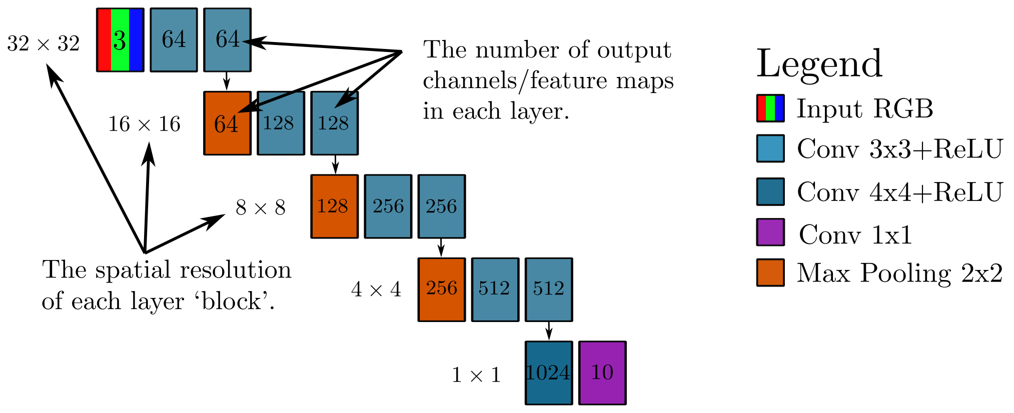


Figure 2.6: An example CNN architecture for image classification. In this case the image is classified into 10 classes as can be seen from the number of channels in the final layer. In this diagram the vertical arrangement groups layers by their spatial resolution which is a convenient standard when illustrating semantic segmentation CNNs later.

Convolutional filters in a CNN are stacked or 'layered' on top of each other. A common architectural choice is to increase the number of convolutional feature maps in later layers. Intuitively the idea behind this is that more numerous complex global image features in higher layers are composed from a hierarchy of less numerous lower-level features such as edges and blobs. Figure 2.7 visualises this hierarchy using the approach developed by [Yosinski et al., 2015]. It operates by optimising image features (subject to certain regularisation constraints) to maximise the activation for a given output feature pixel. Simple edges and colour features can be seen in the bottom layer which are composed in later layers to larger and more complex features. The final layer exhibits recognisable characteristics of the associated object.

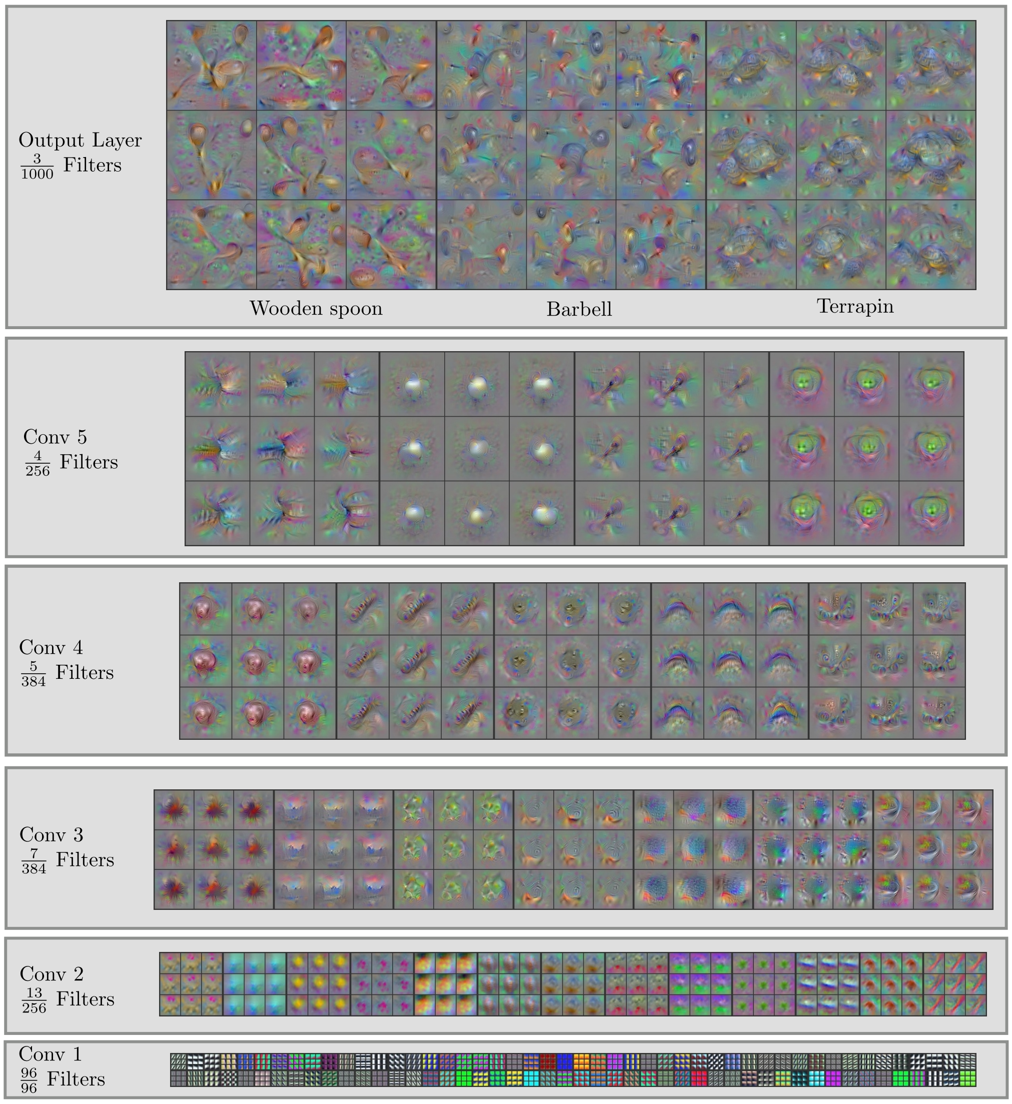


Figure 2.7: A visualisation of the hierarchy of features at different layers of an ‘AlexNet’-type CNN trained on ImageNet (ILSVRC 2012). The size of the images above indicate the receptive field of the filter, and for each filter the output of 9 different gradient descent runs are visualised. Created using optimised images and source code from `https://github.com/yosinski/deep-visualization-toolbox`.

The trainable parameters within these convolutional filters are known as the

CNN's 'weights.' Supervised training proceeds by updating the weights using gradient descent methods in all of the layers of the CNN via backpropagation, discussed in Section 2.5.6. The supervisory signal to compute the gradient is provided by training examples with accompanying ground truth labels. The first example of this style of modern CNN architecture and training method was proposed by [LeCun et al., 1998] and was called 'LeNet-5' which was used for character recognition on the MNIST dataset of handwritten digits. A useful resource for further reading on Deep Learning in general is the textbook [Goodfellow et al., 2016].

2.5.1 Semantic Segmentation

In later chapters CNNs are used for both object detection and instance segmentation, however the most common task CNNs are used for in this thesis is semantic segmentation. This can be thought of as a simple generalisation of the classical image classification task. In image classification the CNN produces a probability distribution describing the probability that a given input image is labelled as belonging to one of a set of n predefined classes, $\mathcal{C} = \{\text{class}_1, \text{class}_2, ...\text{class}_n\}$. The probability distribution over all n classes is denoted as $p(c \mid I)$. A final layer called a 'fully connected' layer is often used for this purpose, which performs a linear mapping on the entire feature map of the layer below it with n equally sized sets of weights via dense matrix multiplication.

The output vector, denoted $\mathbf{z} \in \mathbb{R}^n$, of the fully connected layer is not a valid probability distribution. A special non-linearity called a 'Softmax' can be used to map the vector of arbitrary real numbers (positive, negative, or zero) to a vector of scalars in the range (0,1) which sum to 1:

$$\mathbf{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum\limits_{j=1}^{n} e^{z_j}}. \tag{2.43}$$

The softmax function has the advantage of providing smooth gradients everywhere which is useful for training and is directly motivated by the maximum likelihood estimation of $-\log(p(c|I))$ and the cross-entropy loss discussed in Section 2.5.6 (see [Goodfellow et al., 2016] for a more in-depth discussion). The outputs of the CNN can be seen as predicting the probabilities in log space and are termed 'logits'. Logits can then be mapped via the softmax back to a valid probability distribution.

Architecturally, a classification output can be achieved by designing the final layer of the CNN to output a tensor of 'logits' of size $n \times 1 \times 1$ where each channel represents a given class and only a single pixel is output to represent a label for the whole image. Although conventionally this dense multiplication is called a fully connected layer, it is convenient here to understand it as a special case of a convolutional layer, where the kernel size exactly matches the input size. In this arrangement each of the n output channels take as input the complete set of feature maps of the layer below, and multiplies each pixel of the input feature map with a learned weight to produce a single activation, the classification logit. This $n \times 1 \times 1$ output for classification has already been illustrated for the 10-class case in the CNN diagram shown in Figure 2.6.

Ensuring every element in the CNN is a convolution is known as a Fully Convolutional Neural Network [Long et al., 2015], and it is particularly useful for semantic segmentation. In semantic segmentation the aim is to output a label for each pixel in a given image rather than one label for the image as a whole. The Fully Convolutional structure outlined above allows the final 'classifier' filter to stride along a larger input feature map to output logits for each pixel in the feature map. The final softmax layer operates independently over the n channels to produce a probability distribution at each pixel location. This probability 'heatmap' over an image still contains n channels for each class, and the argmax is used to select the most likely class as shown in Figure 2.8.

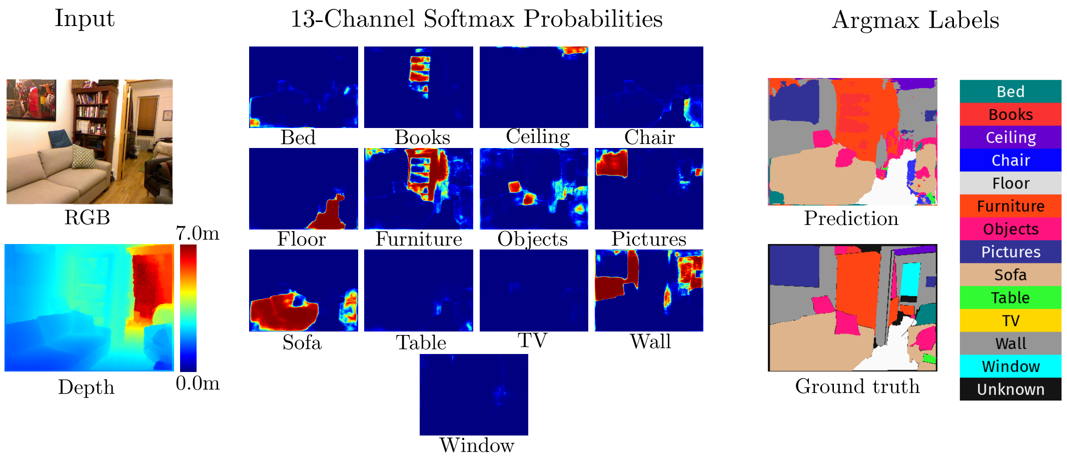


Figure 2.8: An example from the NYUv2 test set of per-channel semantic probability maps as well as the final labels compared to the ground truth labelling.

As already discussed, a common architectural choice in a CNN is to reduce the spatial resolution in later layers and increase the number of feature maps. However this creates a problem for semantic segmentation which also requires a high spatial resolution in the final prediction layer. A wide range of approaches to this problem exist [Long et al., 2015, Noh et al., 2015, Ronneberger et al., 2015, Lin et al., 2017, Chen et al., 2018], but almost all involve some form of encoder-decoder style bottleneck architecture alongside some form of 'skip connections.'

The bottleneck approach is designed around a normal CNN with progressively lower spatial resolution and larger numbers of feature maps up to a minimum spatial resolution known as the 'bottleneck.' After this, the operations are reversed in a decoder, with progressively larger spatial resolutions (via classical or learned upsampling methods) and fewer feature maps. This allows global image information to be combined in the bottleneck to condition the decoder output, but fine-grained spatial resolution has been lost.

To maintain spatial information skip connections duplicate feature maps of high resolution encoder layers and provide them as input to the upsampled decoder layers. This can be supplied as input either through concatenation (stacking feature maps channel-wise) such as in U-Net [Ronneberger et al., 2015] or via element-wise addition of the feature map activations such as in [Long et al., 2015]. Many alternative approaches have also been used. For example in Section 4.3.2 an architecture which stores max pooling indices and 'unpools' them later is used [Noh et al., 2015]. Figure 2.9 provides a network diagram of an example U-Net architecture.

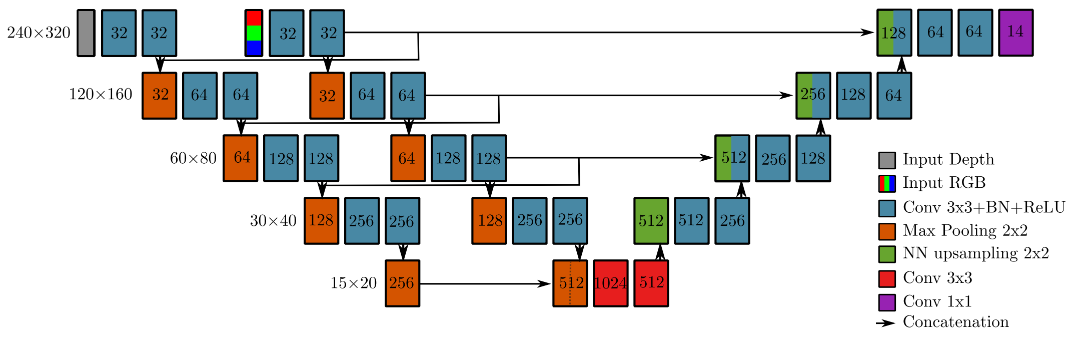


Figure 2.9: A example network diagram. In this case showing the RGB-D architecture based on U-Net used in Chapter 5 for semantic segmentation.

Although the above architecture is a useful starting point, and is used throughout the thesis it must be noted that it is quite limited in the number and generality of the classes provided. Throughout this work somewhat arbitrarily defined sets of common indoor semantic classes are used for training and prediction. It is of course possible to encode a large number of classes in the manner above, however many of these class tokens may be closely related and even form a hierarchy. It is therefore unsatisfactory to treat the mistake between say a 'monitor' and a 'television' as equally incorrect as a 'cat' from a 'ceiling'.

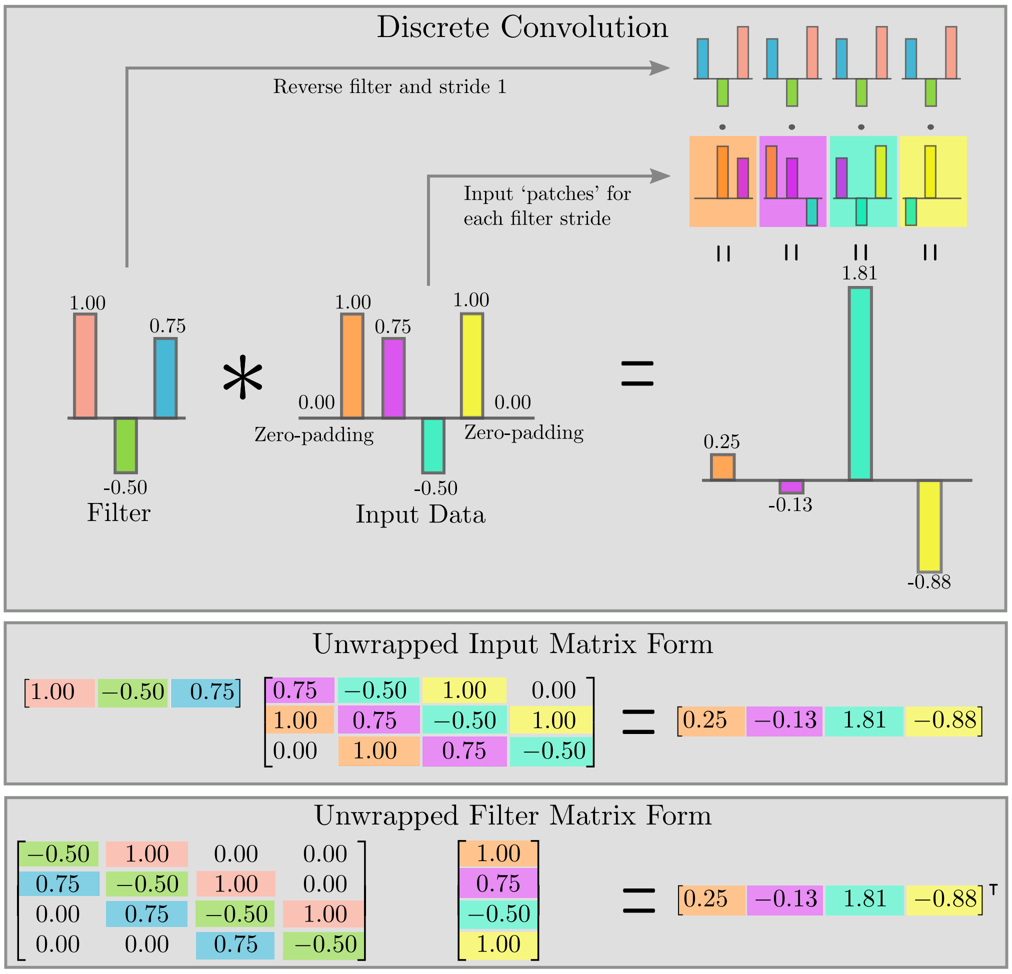
This property of semantic similarity has been studied extensively in the machine learning field called Natural Language Processing. It has been known since the early 2000s [Bengio et al., 2001] that semantically meaningful low-dimensional ($\approx$300) vector spaces can be learned rather than using arbitrary 'one-hot' tokens with dimension equal to the number of words. In this way similar words such as capital cities, animals, people, and verbs are all grouped in close proximity. Furthermore, vector addition illustrates particularly interesting semantic operations such as (England-London)$\approx$(Spain-Madrid) and (King-Man)$\approx$(Queen-Woman) [Mikolov et al., 2013].

Replacing classification with regression in this semantic space, such as in [Socher et al., 2013], would not only provide a more nuanced and semantically error-weighted cost function, but also allow for novel guesses and ambiguity in responses. If the CNN cannot classify an animal based on its appearance, the cost of guessing a vector in the vicinity of 'animal' will be less costly than a completely random guess and also still provides some useful information to the user. This is an interesting area for future study.

2.5.2 Discrete Convolutions

The core operation in a CNN is of course the convolution. A convolution is the integral of the product of two functions, after flipping one (here called the 'filter') and striding it along the other. In discrete convolutions it is the sum over the area of support of the kernel and the input, as shown in Figure 2.10. It can also be seen from Figure 2.10 that in the discrete case, the product operation is the dot product, and unwrapping the convolutional filter (or alternatively the input) into a matrix form can reduce the convolution operation to simple matrix multiplication.

In most Deep Learning frameworks the reversal of the convolutional filter is in fact omitted (or assumed to be performed prior to the operation) and so they actually are performing an operation related to a convolution called 'cross-correlation.' As the filters are learned from randomly initialised weights in any case, this omission does not have a significant impact. In 2D, the filter strides over an input tensor


Discrete Convolution
Reverse filter and stride 1
Input 'patches' for each filter stride
1.00
0.75
-0.50
Filter
1.00
0.75
1.00
0.00
0.00
Zero-padding
Zero-padding
-0.50
Input Data
1.81
0.25
-0.13
-0.88
Unwrapped Input Matrix Form
Unwrapped Filter Matrix Form

$$\begin{bmatrix} 1.00 & -0.50 & 0.75 \end{bmatrix} \begin{bmatrix} 0.75 & -0.50 & 1.00 & 0.00 \\ 1.00 & 0.75 & -0.50 & 1.00 \\ 0.00 & 1.00 & 0.75 & -0.50 \end{bmatrix} = \begin{bmatrix} 0.25 & -0.13 & 1.81 & -0.88 \end{bmatrix}$$

$$\begin{bmatrix} -0.50 & 1.00 & 0.00 & 0.00 \\ 0.75 & -0.50 & 1.00 & 0.00 \\ 0.00 & 0.75 & -0.50 & 1.00 \\ 0.00 & 0.00 & 0.75 & -0.50 \end{bmatrix} \begin{bmatrix} 1.00 \\ 0.75 \\ -0.50 \\ 1.00 \end{bmatrix} = \begin{bmatrix} 0.25 & -0.13 & 1.81 & -0.88 \end{bmatrix}^{\intercal}$$

Figure 2.10: A discrete convolution (in 1D for clarity). A kernel of size 3 is applied to a 4-length input. Zero-padding of 1 ensures the output length is also 4. The Toeplitz matrix forms for both unwrapped input and filter are shown below.

which can be either an image or the output of previous convolutional operations (the feature map). As in the 1D case, the filter slides along the input by a number of pixels known as the 'stride.' The spatial dimension of the convolutional filter (or 'kernel') is called the 'kernel size' which for a single channel in 2D is a $H \times W$ matrix, where H denotes kernel height, W kernel width. A common kernel size is $H = W = 3$. The filter itself is actually of size $C \times H \times W$ where C is dictated by the number of input channels (*e.g.* for RGB it would be $C = 3$).

To ensure a spatial output size equal to the input size, the convolutional stride is often set to 1 and zero-padding of $\frac{\text{kernel size}-1}{2}$ is added to the input (for the common 3×3 kernel size zero-padding of size 1 is added). For each stride of a convolution a single output activation is calculated. An additional learned ‘bias’ parameter is also generally added to the filter’s final activation, and can be viewed as another weight operating on a constant pseudo-input of 1. This process is visualised in Figure 2.11 for a single filter with a 3-channel input. In a manner analogous to the 1D case, the output of this convolution also can be seen as simple matrix multiplication by appropriate unwrapping of the filters. This unwrapping is explicitly shown in a small example when discussing transpose convolutions in Figure 2.14.

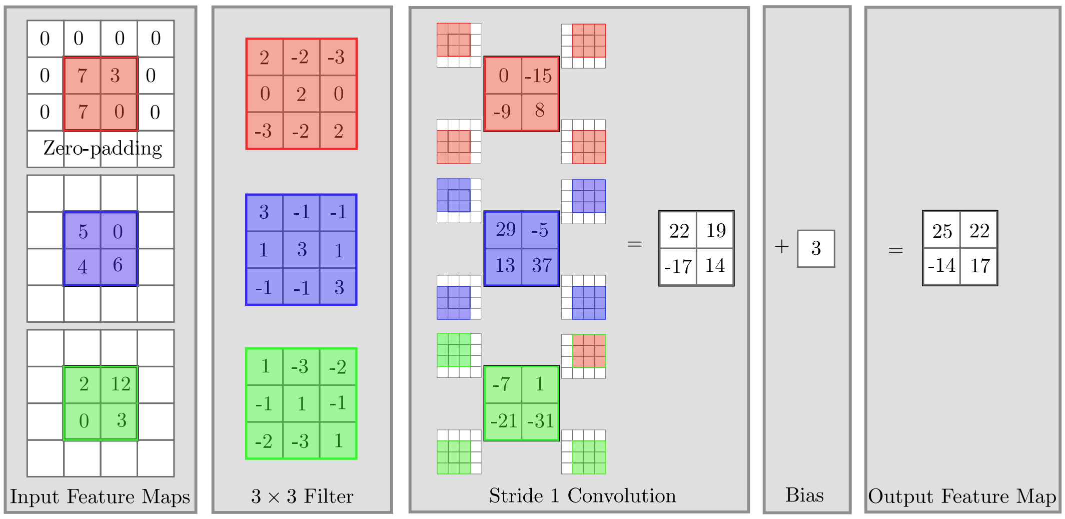


Figure 2.11: Discrete 2D image convolutions with a single filter (and hence, single output channel) on an RGB input image.

Essential components within a CNN are the non-linearities or ‘activation functions.’ As convolutions can be viewed as matrix multiplication, repeated composition without any non-linearities means the entire network can be compacted to a single linear operation on the inputs. Including a non-linearity after each convolutional layer allows for more expressive non-linear functions to be composed. This non-linearity is applied elementwise to the feature map. Many non-linearities are in use, but in this thesis the most commonly used is the Rectified Linear Unit or ‘ReLU.’ It is a simple piecewise linear function defined as,

$$\mathrm{ReLU}(x) = \max(0, x), \tag{2.44}$$

and is shown alongside other common activation functions in Figure 2.12.

A given ‘layer’ in a CNN consists of a number of filters, each of which processes the same input to produce one of the output channels. For example, a $3 \times 480 \times 640$ RGB input image has 3 output channels. This would require a $3 \times 3 \times 3$ kernel

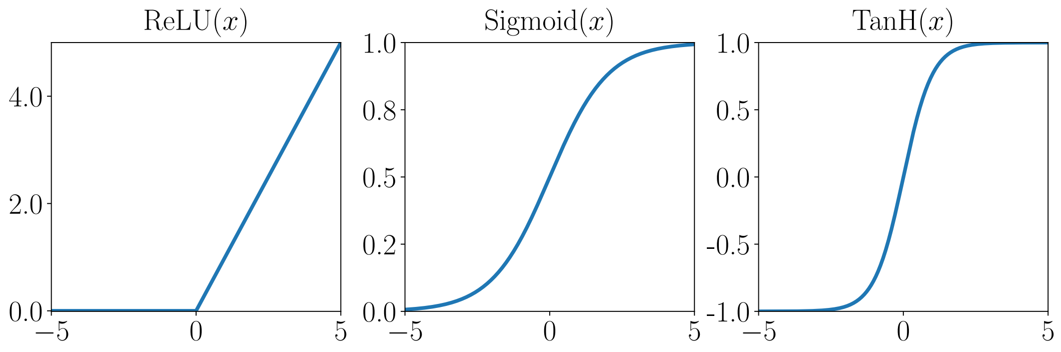


Figure 2.12: The ReLU function alongside other common non-linearities.

size (if using the common 3×3 kernel), with a zero-padding of 1. If there are 64 of these filters they will output a $64 \times 480 \times 640$ feature map that goes through the non-linearity function which is the output of this layer and passed into the next layer. The next set of 3×3 convolutional filters will then require kernels of size $64 \times 3 \times 3$.

2.5.3 Max Pooling

If feature maps were left at full image resolution in later layers it would result in enormous computational and memory requirements. Therefore it is also a common feature that the spatial dimension in a CNN decreases as the number of feature maps increases. Simply increasing the stride of a convolution as already described is one way to reduce the spatial dimension of a feature map, but a more commonly used approach to down sample a feature map is called Max Pooling.

Max Pooling is designed to reduce the spatial dimensions of a feature map while preserving the highest activations. It is frequently included in CNNs after a ‘block’ of similarly sized convolutional layers. The idea is to pool multiple input activations (on each feature map independently) into a single output by taking the maximum activation, as depicted in Figure 2.13. The kernel size and stride can vary, but the CNNs in this thesis almost exclusively perform max pooling with a stride of 2 and kernel size of 2. This has the effect of halving the input size if the feature map spatial dimensions are a multiple of 2, such that a $64 \times 256 \times 256$ feature map would be reduced to a $64 \times 128 \times 128$ feature map.

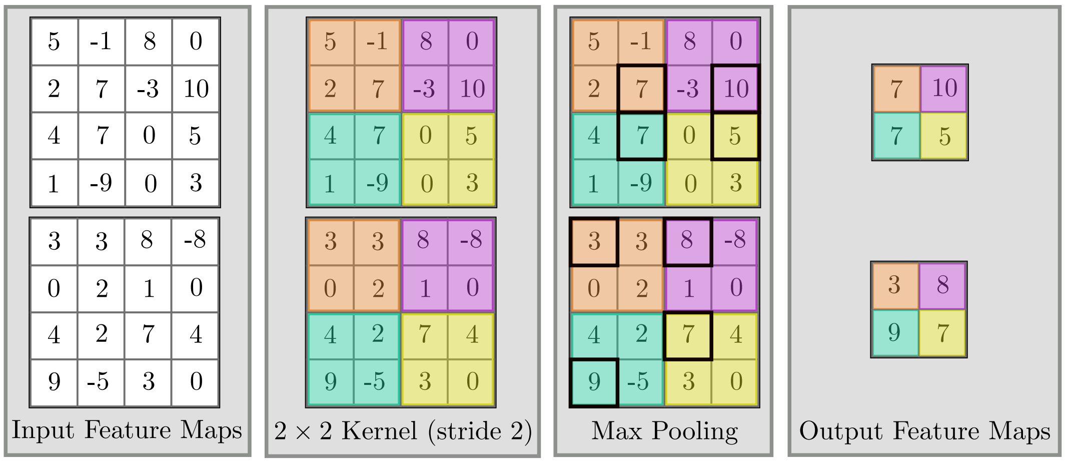


Figure 2.13: An illustration of the max pooling operation on two feature maps.

2.5.4 Batch Normalisation

Although there are a number of other layers which will be used within this thesis, a common addition to the standard convolutional layer described above is Batch Normalisation [Ioffe and Szegedy, 2015]. For semantic segmentation we have found Batch Normalisation to be a useful addition. It is particularly helpful during training as it reduces the number of training steps required for convergence, but it does lead to complications during inference. It is also a foundational building block in many state-of-the-art CNNs such as the Residual Network (ResNet) architecture [He et al., 2016].

Batch Normalisation was developed to mitigate the changing activation distributions input to upper layers in a CNN as the weights in lower layers are updated. In [Ioffe and Szegedy, 2015] they speculate on this issue (termed *internal covariate shift*) and propose adding a normalisation layer to address it. In the original work, and here, Batch Normalisation is situated between the convolutional layer and the non-linearity, with a complete 'layer' often abbreviated to `Conv+BN+ReLU`. The alternative ordering (applying Batch Normalisation after the ReLU) has also been evaluated and found to be beneficial in certain architectures [Mishkin et al., 2016].

The Batch Normalisation layer operates by subtracting the mean and dividing by the standard deviation of individual mini-batches during training (as discussed in Section 2.5.6) and is applied per *filter kernel*. Grouping by filter kernel means that all of the output activations produced from a given filter are included in the statistics, i.e. all of the output pixels in all of the training examples in the mini-batch. Various other grouping schemes for calculating statistics have also (almost exhaustively) been proposed [Ba et al., 2016, Ulyanov et al., 2016b, Wu and He, 2018].

To allow this normalised output to also represent the identity transform and to provide generality, two parameters are included which operate to scale (γ) and bias (β) the normalised value. These parameters are differentiable with respect to the loss and so are also trained alongside other weights in the CNN. They are commonly initialised with $\gamma = 1$ and $\beta = 0$.

The calculation of statistics based on mini-batches allows for efficient training, however during inference there is only a single example, and it may not be desirable to operate on the statistics of that sample. In [Ioffe and Szegedy, 2015] the exponential moving average of the mean and variance statistics of the previous mini-batches are calculated and stored. As the weights of lower layers, and hence input values to upper layers are changing it is useful to slowly decay the previous statistics by a constant τ (commonly set to 0.999) as older activations become less representative of the activations to be expected during inference. During inference these stored statistics operate on the feature maps rather than statistics calculated for the current example. Algorithm 1 shows the operation of the BatchNorm layer on a single filters' output.

Algorithm 1: Batch Normalisation

```
Input      : Filter activations $x_{1..N} \in \mathcal{X}$.
             Moving average statistics $EMA_{\mu}$ and $EMA_{\sigma^2}$.
Parameters: Learned scale $\gamma$ and bias $\beta$. Moving average decay, $\tau$.
Output     : Output activations $z_{1..N} = \mathrm{BN}_{\gamma,\beta}(x_{1..N})$
1 $\mu \leftarrow \frac{1}{N}\sum_{i=1}^{N} x_i$;
2 $\sigma^2 \leftarrow \frac{1}{N}\sum_{i=1}^{N} (x_i - \mu)^2$;
3 $\mathrm{EMA}_{\mu} \leftarrow \tau\mathrm{EMA}_{\mu} + (1-\tau)\mu$;
4 $\mathrm{EMA}_{\sigma^2} \leftarrow \tau\mathrm{EMA}_{\sigma^2} + (1-\tau)\sigma^2$;
5 for $i \leftarrow 1$ to $N$ do
6     $x_i' \leftarrow \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$;
7     $z_i \leftarrow \gamma x_i' + \beta$;
8 end
```

2.5.5 Upsampling

It is necessary to increase the size of feature maps within a CNN in order to achieve a reasonable resolution for tasks such as semantic segmentation. A number of approaches to this problem exist, but the two main techniques used in this thesis are learnable 'transpose convolutional' layers and non-learned image upsampling. The first approach is to learn the weights of a convolution which increases the feature map size. This is accomplished by transposing the filter matrix illustrated in Figure 2.10, which has the result of multiplying a larger kernel by the input activation and cumulating the strided output of this kernel into an output feature map.

Figure 2.14 shows the operation of the transpose convolutional layer, and it also highlights a number of important caveats about the operation. It shows how transposing the convolutional filter matrix allows one to invert the original operation in terms of the sizes of the feature maps (as long as the matrices are appropriately reshaped). It also shows that the operation is *not* the inverse of a convolution, as performing the operation on the output with the same filter does not result in the original input again. This is the reason why the widely used term for a transpose convolution of 'Deconvolution' is actually quite misleading.

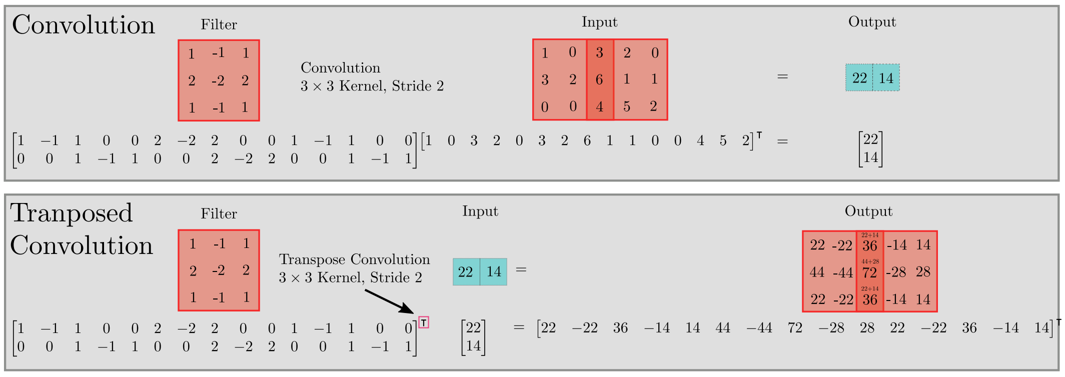


Figure 2.14: A diagram showing the operation of a transpose convolutional layer with its equivalent matrix form.

A frequently encountered issue with the transpose convolutional layer is that the cumulative striding causes a checkerboard pattern. This effect is also illustrated in Figure 2.14 where the output values on the intersection of the two strided filters can be seen to be larger than the non-intersected areas. For this reason [Odena et al., 2016] suggest using image upsampling instead of transpose convolutional layers. These upsampling layers can then be followed by a normal convolutional layer. In this thesis nearest neighbour upsampling is the most common choice, however some CNNs also use bilinear upsampling, and this will be specified where appropriate.

2.5.6 Optimisation and Cross-Entropy Loss

The weights of a CNN are optimised (also often described as 'trained' or 'learned') using gradient descent to minimise a selected loss function, denoted L, on a given dataset. A large variety of numerical optimisation procedures are used by the Deep Learning community,[2] however in this thesis the majority of CNNs are optimised using the very simple and standard Stochastic Gradient Descent algorithm with momentum (SGD). Momentum acts like its physical analogue to smooth noisy gradients and better move past saddle points and small bumps in the loss function by accumulating past gradients into the velocity variable, $\mathbf{v}$. For weights, $\mathbf{w}_t$, at training update index t, updates proceed according to the scheme:

$$\mathbf{v}_{t+1} = \mu \mathbf{v}_t - \alpha \frac{\partial L}{\partial \mathbf{w}_t} \tag{2.45}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_{t+1}. \tag{2.46}$$

Here α is called the 'learning rate' and scales the gradient. It is commonly initialised to 0.01 and scaled by a factor of 0.1 according to a schedule based on training iterations or loss convergence. The variable μ specifies the momentum exponential decay, and is often set to 0.9. A μ of 0 implies no momentum and a μ of 1 means previous updates are cumulated indefinitely. For robustness the output gradients are also sometimes L2-norm clipped with respect to a threshold τ, often set to 5. When the gradient's L2-norm exceeds that threshold it is rescaled by a factor $\frac{\tau}{|\frac{\partial L}{\partial \mathbf{w}}|_2}$.

The 'Stochastic' term in Stochastic Gradient Descent stems from calculating the gradients based on a single sample of the full dataset as an approximation to the true gradient. In practice small 'mini-batches' of samples are used to reduce the noise in parameter updates and for computational efficiency. For this reason, mini-batch SGD is sometimes called 'Mini-batch Gradient Descent,' but here we will continue to use the term SGD for mini-batches also. The full list of training samples is randomly shuffled and samples are removed from the list in mini-batches. After all of the training samples in the dataset have been used (termed an 'epoch') the dataset is shuffled again and the procedure repeats.

[2]Examples include Adadelta, Adagrad, Adam, BFGS, and RMSProp to name but a few.

As the gradient is required to train the CNN, the chosen loss function must be differentiable and penalise predictions far from the ground truth label. A common loss function for classification, and by far the most frequently used in this thesis, is the negative log-likelihood of the multinomial distribution also known as the cross entropy loss. Cross entropy, denoted $H(p, q)$, is a term derived from information theory which calculates the expected 'cost' in bits (or 'nats' if using the natural logarithm) of using a probability distribution, q, to design an optimal encoding scheme to specify which sample, such as a class label c from a set of n classes, was drawn from the true distribution of labels, p, in a random trial:

$$H(p, q) = -\sum_{i=1}^{n} p(c_i) \log q(c_i). \tag{2.47}$$

In our classification task, training examples are pairs of images and labels, (I, g), where labels can be represented with a distribution, $p(c)$, and are encoded as a 'one-hot' vector, with a probability of 1 for the ground truth class, c_g, and 0 elsewhere. It can be seen from Equation 2.47 that in this case the result of the summation is only dependent on one value of the $q(c)$ distribution, namely $q(c_g)$ i.e. the probability the CNNs assign to the true ground truth class (although the other logits do often have an implicit impact due to the normalisation in the Softmax function below). The loss for a given training example therefore becomes the negative logarithm of the predicted probability of the ground truth class, which is 0 when $q(c_g) = 1$ and approaches ∞ as $q(c_g)$ tends towards 0. A plot is shown in Figure 2.15.

The probability distribution $q(c)$ is the output of the last CNN layer and can be seen as a function $\mathbf{f}(I, \mathbf{w})$ of the CNN weights, $\mathbf{w}$, and the input image, I, which then goes through the softmax to produce a probability distribution. For a number of training examples, M, such as a mini-batch or the set of all pixels in a mini-batch for semantic segmentation, the loss is calculated by taking the average:

$$L(I, g, \mathbf{w}) = -\frac{1}{M} \sum_{m=1}^{M} \log \mathbf{softmax}(\mathbf{f}(I_m, \mathbf{w}))_{g_m}. \tag{2.48}$$

A regulariser on all of the weights, $\mathbf{w}$, such as an L2-norm is also commonly used to prevent over fitting via large weights. This results in an extra term, $\frac{1}{2}||\mathbf{w}||_2^2$, being added to the standard classification loss function.

Before training the weights are often initialised with small random values either from a Gaussian distribution, a more refined scheme such as Glorot [Glorot and

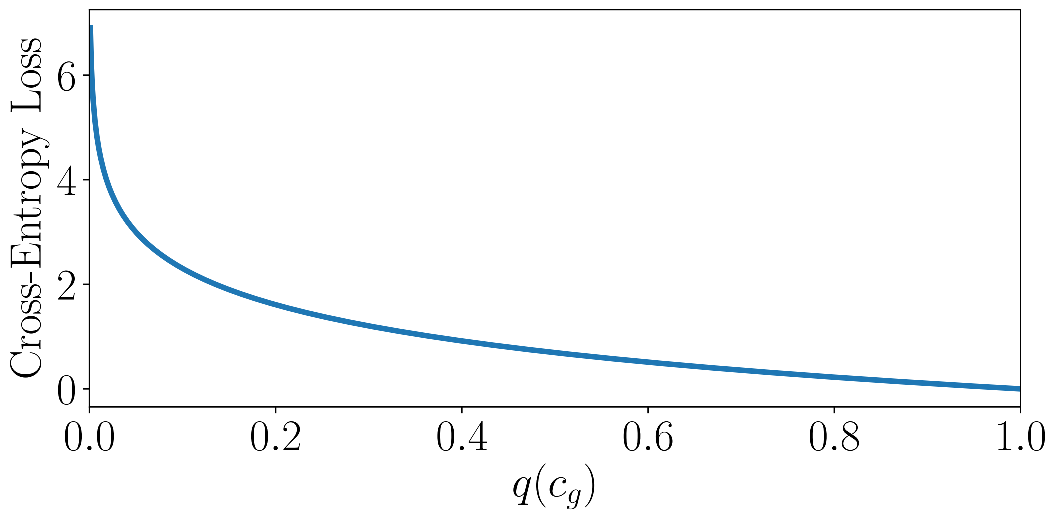


Figure 2.15: The Cross-Entropy or Negative Log-Likelihood Loss

Bengio, 2010] or He initialisation [He et al., 2015]. After that, a training example is fed into the network during the forward pass (or forward-propagation), and the output prediction is used to calculate the scalar loss. To update the weights, back-propagation [Rumelhart et al., 1986] is used to calculate the gradients of the loss with respect to each of the weights for a given training example.

As we operate on K sequential layers of the CNN, the function $\mathbf{f}(I_m, \mathbf{w})$ can be viewed as a composition of functions, with each function representing a layer:

$$\mathbf{f}(I, \mathbf{w}) = \mathbf{f}_K(..\mathbf{f}_2(\mathbf{f}_1(I, \mathbf{w}_1), \mathbf{w}_2)..\mathbf{w}_K). \tag{2.49}$$

Although a 'layer' was above described in terms of CNN architectures as the composition of a convolution and non-linearity (sometimes with Batch Normalisation), for the purposes of back-propagation these operations are separated to produce more modular systems and simpler derivative calculations. Operations are occasionally combined again for numerical stability, such as when taking the derivative of the commonly combined softmax and cross entropy layers. Back-propagation is essentially a recursive application of the chain rule to the composite CNN function and the loss function. It works backwards starting from the derivative of the scalar loss with respect to the output of the final CNN layer:

$$\frac{\partial L}{\partial \mathbf{f}_K} = \frac{1}{M} \sum_{m=1}^{M} \mathbf{softmax}(\mathbf{f}(I_m, \mathbf{w})) - 1. \tag{2.50}$$

After calculating the $\frac{\partial L}{\partial \mathbf{f}_K}$ term for the first layer, this term can be used to calculate the $\frac{\partial L}{\partial \mathbf{w}_K}$ term needed for the $\mathbf{w}_K$ weight update. This is possible because, by design, all of the layer functions are differentiable with respect to their weights. It can also be used to calculate the loss with respect to the inputs to the layer, $\frac{\partial L}{\partial \mathbf{f}_{K-1}}$:

$$\frac{\partial L}{\partial \mathbf{w}_K} = \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{w}_K}, \tag{2.51}$$

$$\frac{\partial L}{\partial \mathbf{f}_{K-1}} = \frac{\partial L}{\partial \mathbf{f}_K} \frac{\partial \mathbf{f}_K}{\partial \mathbf{f}_{K-1}}. \tag{2.52}$$

The $\frac{\partial L}{\partial \mathbf{f}_{K-1}}$ term can then be passed down to the $K-1$ layer and is used to calculate the weight gradients of the layer below as well as the $\frac{\partial L}{\partial \mathbf{f}_{K-2}}$ term for the layer after that and so on:

$$\frac{\partial L}{\partial \mathbf{w}_{K-1}} = \frac{\partial L}{\partial \mathbf{f}_{K-1}} \frac{\partial \mathbf{f}_{K-1}}{\partial \mathbf{w}_{K-1}}, \tag{2.53}$$

$$\frac{\partial L}{\partial \mathbf{f}_{K-2}} = \frac{\partial L}{\partial \mathbf{f}_{K-1}} \frac{\partial \mathbf{f}_{K-1}}{\partial \mathbf{f}_{K-2}}. \tag{2.54}$$

In practice the full Jacobians $\frac{\partial \mathbf{f}_{K-1}}{\partial \mathbf{w}_{K-1}}$ and $\frac{\partial \mathbf{f}_{K-1}}{\partial \mathbf{f}_{K-2}}$ are not explicitly calculated and multiplied with the $\frac{\partial L}{\partial \mathbf{f}_{K-1}}$ Jacobian, because this would require huge amounts of memory and be wasteful in the case of sparse connections such as elementwise non-linearities or shared weights in a convolution. Instead a direct product and sum aggregation of the required partial derivatives, $\frac{\partial L}{\partial \mathbf{f}_{K-1}}$, with the relevant weights, $\mathbf{w}_{K-1}$, and inputs, $\mathbf{f}_{K-2}$, can often be performed.

Additionally, although simplified here in the mathematical notation to vectors and Jacobian matrices, in practice 4-dimensional tensors are required for training a 2D CNN (batch size $\times$ channels $\times$ height $\times$ width). However these can also be thought of as being 'reshaped' into an appropriate vector form and this does not affect the above mathematical underpinnings.

After calculating the gradients for all of the weights in the CNN they are updated simultaneously according to Equation 2.45. The updated weights are then used for the forward pass of the next mini-batch of training data. This procedure continues until some stopping criteria is reached, often after a set number of training epochs or when a chosen error metric begins to increase on a held-out validation set.

Other more complicated tasks than classification are approached using CNNs in later chapters of this thesis, such as object detection and instance segmentation. The 'backbone' of these architectures rely heavily on the layers and methods already described in this chapter, but generally include some additional task specific operations and custom loss functions. These will be introduced and described in more detail in the relevant section.

2.5.7 Evaluation

During optimisation (and afterwards) performance metrics provide a valuable diagnostic tool to evaluate the improvement or detriment of a change in the methodology. A number of different performance metrics are used in this thesis to quantitatively evaluate the semantic segmentation performance. The simplest metric is the pixel (or global) accuracy. That is, for each pixel with a valid ground truth labelling the predicted label is compared and the proportion of correctly assigned labels out of all valid ground truth labels is the accuracy. Using N_{gp} to denote the number of pixels with ground truth label $g \in \mathcal{C}$ which have been predicted to be class $p \in \mathcal{C}$:

$$\text{Pixel Accuracy} = \frac{\sum\limits_{g \in \mathcal{C}} N_{gg}}{\sum\limits_{g \in \mathcal{C}, p \in \mathcal{C}} N_{gp}}, \tag{2.55}$$

for example, if there are 1000 labelled pixels ($\sum\limits_{g \in \mathcal{C}, p \in \mathcal{C}} N_{gp} = 1000$), and for 600 of them the predictions are correct, ($\sum\limits_{g \in \mathcal{C}} N_{gg} = 600$), then the pixel-wise accuracy is 0.60.

A complication commonly encountered is that labelled images often contain areas where no labelling has been provided called a 'void' or 'unknown' class. For the purposes of most metrics these pixels are completely discounted from any performance metric as one cannot tell whether the system's prediction was correct or not. As the performance metrics described here are defined at the pixel-level this implies that images with large areas that are not labelled count for less in the final total.

For diagnostic purposes, summing the performance of a system into a single number can mask problems. A confusion matrix is a useful depiction of all of the misclassifications for a multi-class segmentation problem. It is a square $|\mathcal{C}| \times |\mathcal{C}|$ matrix in which each entry is the already defined term, N_{gp}. Dividing each row of a confusion matrix by the sum of that row ($\frac{N_{gp}}{\sum\limits_{g \in G} N_{gp}}$) produces a normalised confusion matrix, in which each entry indicates the proportion of ground truth pixels classified as a certain other class. From the perspective of the confusion matrix, the pixel-wise accuracy is the sum of the diagonal (the trace) over the sum of the entire matrix.

When there are imbalanced classes the simple pixel-wise accuracy can be quite misleading. For example suppose there are only two classes, $\mathcal{C} = \{\text{structure}, \text{books}\}$. If structure makes up $\frac{99}{100}$ pixels of the ground truth then a semantic predictor which always predicts 'structure' will achieve 99% Pixel Accuracy without even looking at the image input. Two common metrics which aim to alleviate this are the Class Average Accuracy and the mean Intersection over Union (mIoU).

The Average Class Accuracy calculates the accuracy for each class independently and then takes the average of them:

$$\text{Average Class Accuracy} = \frac{1}{|\mathcal{C}|} \sum_{g \in \mathcal{C}} \frac{N_{gg}}{\sum\limits_{p \in \mathcal{C}} N_{gp}}. \tag{2.56}$$

This is equivalent to the mean of the diagonal of the normalised confusion matrix. In the case of the 'structure' predictor described above the average class accuracy is therefore $\frac{1}{2}(\frac{99}{99}+\frac{0}{1})$=50%, as it achieves 100% for the structure class but 0% for books. This is a fairer sounding statistic than 99% however as noted by [Everingham et al., 2010], giving an individual class accuracy of 100% for structure is itself misleading. If the classifier predicts a certain pixel to be structure, even on this dataset in which it scored 100% accuracy for structure it still only has a 99% probability being correct, as 1% of the time it produced a false positive.

The mIoU metric is similar to the Average Class Accuracy, except that it takes into account false positives in the denominator:

$$\text{Mean Intersection over Union} = \frac{1}{|\mathcal{C}|} \sum_{g \in \mathcal{C}} \frac{N_{gg}}{\sum\limits_{p \in \mathcal{C}} N_{gp} + \sum\limits_{g' \in \mathcal{C}} N_{g'g} - N_{gg}}. \tag{2.57}$$

As the name suggests, this is done for each class by taking the area of correct predictions (the areas where the predictions class overlaps with the ground truth) and dividing by the union of these two areas (i.e. the sum of both areas minus the intersection). In the case of the structure predictor the IoU for structure is $\frac{99}{99+100-99} = 99\%$ and $\frac{0}{1+0-0} = 0\%$ for floor giving a mIoU of 49.5%. Figure 2.16 illustrates a worked example of all of the described metrics, as well as the equivalent confusion matrix from which they can be calculated for a given ground truth label and prediction.

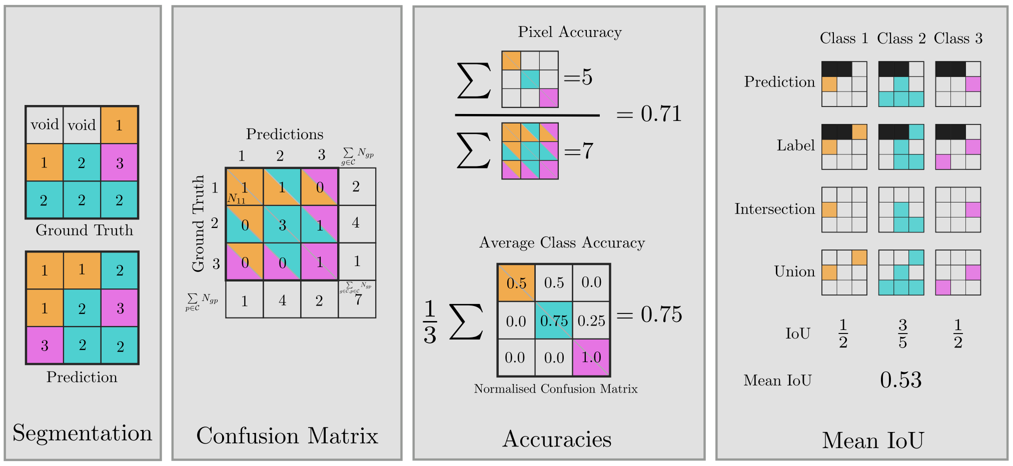


Figure 2.16: An illustration of the frequently used semantic segmentation performance metrics.

It should be noted that for specific tasks certain misclassifications could be considered much more costly than others. In such a case more specialised metrics should be used to account for the cost of a given error. In this work the more generic performance metrics commonly used in semantic segmentation benchmarks is often used. The idea being to provide a general semantic mapping framework which can be tailored to specific tasks when required.

2.6 General-Purpose Graphics Processing Units

The two main threads of research in this thesis of Deep Learning and dense SLAM have both been enabled by the rise of powerful consumer Graphics Processing Units (GPUs). A very brief introduction to GPU computing is therefore provided here and some more specific implementation details will be included in later chapters where appropriate.

GPUs were, as the name suggests, originally designed for 3D rendering and were optimised for performing the same set of operations on large batches of input data. They have thousands of processing cores in contrast to modern Central Processing Units (CPUs) which have tens of cores. A GPU is designed to operate in a highly parallel manner with multiple Stream Processors containing local memory on which their many cores can operate quickly and a larger slower DRAM memory which is global to the whole GPU is used for memory transfers to and from the host CPU.

The allocation of more die space to Arithmetic Logic Units on a GPU provides greater bulk computational processing power than the CPU.[3] In previous years, accessing this computational power relied on harnessing specialised graphical shader pipelines in a manner they were not originally designed for in order to solve the actual computational problem at hand. The availability of higher level General-Purpose GPU programming languages such as CUDA and OpenCL compatible with modern consumer GPU hardware has now granted significantly improved access to the computational power of GPUs to the scientific community.

The GPU hardware used in this thesis is manufactured by NVIDIA. NVIDIA has developed the CUDA programming language for programming computation on their GPU hardware. CUDA provides a useful API and compiler (`nvcc`) which can be accessed and called in a straightforward manner in C++ programs. Listing 2.1 shows a simple CUDA program illustrating many of the most basic operations involved in a CUDA kernel. In this case the kernel sums the values in an array, using a reduction algorithm similar to that employed in Chapter 6 for ICP Jacobian reduction of the Gauss-Newton system.

In the program the required global GPU memory is first allocated and the CPU data is copied across to the GPU device memory. The CUDA kernel is launched using the `<<<blocksDim,threadsDim,sharedMemoryBytes>>>` chevrons to indicate how many parallel thread blocks and threads of the kernel should run, as well as how much shared memory should be allocated for each thread block. Shared memory is a faster type of memory than global memory, but is much smaller in size with data that can only be accessed by threads in the same thread block. Within the kernel itself there is a `threadIdx` (and `blockIdx`) to identify a given thread and organise the operations it should perform.

The `sum_reduce` kernel is called with one block of 512 threads, and the `shared_mem` array consists of 512 `float` variables, one for each thread. Each thread initialises its `shared_mem` location (indexed by `threadIdx.x`) to 0, and then cumulates the values from its part of the input data into that location. Its input data is defined in a blockwise fashion, with threads accessing sequential global memory addresses in each iteration of the `for` loop. In the first iteration thread 0 adds `x[0]` and thread 1 adds `x[1]`, up to thread 511 which adds `x[511]`. In the second iteration thread 0

[3] Illustrative diagrams can be found in the NVIDIA Programming Guide [NVIDIA, 2018] available at https://docs.nvidia.com/cuda/cuda-c-programming-guide/.

Listing 2.1: An example C++ CUDA program showing a simple sum reduction kernel.

```cpp
#include <iostream>
#include <vector>

__global__ void sum_reduce(float *x, float *result, int size) {
  // The size of shared memory is set by the launch kernel
  extern __shared__ float shared_mem[];
  // Each thread cumulates part of the input blockwise into shared memory
  shared_mem[threadIdx.x] = 0.0;
  for (unsigned int xid = threadIdx.x; xid < size; xid += blockDim.x)
    shared_mem[threadIdx.x] += x[xid];
  // Ensure all threads have finished cumulation before the final reduction
  __syncthreads();
  // In thread 0 sum the values from all the other thread's shared memory
  if (threadIdx.x == 0) {
    for(unsigned int tid = 1; tid < blockDim.x; ++tid)
        shared_mem[0] += shared_mem[tid];
    // Set the output result equal to the value in thread 0's shared_mem
    result[0] = shared_mem[0];
  }
}

int main() {
  // CPU data with 100,000 length array filled with 7s
  std::vector<float> x(100000,7);
  const int x_bytes = x.size() * sizeof(float);
  // Allocate GPU memory for input data and result
  float *gpu_x, *gpu_result;
  cudaMalloc((void**) &gpu_x, x_bytes);
  cudaMalloc((void**) &gpu_result, sizeof(float));
  // Copy input data from CPU to GPU
  cudaMemcpy(gpu_x, x.data(), x_bytes, cudaMemcpyHostToDevice);
  // Launch kernel on the GPU with number of threads and shared memory size
  const int threads = 512;
  const int shared_mem_size = threads * sizeof(float);
  sum_reduce<<<1,threads,shared_mem_size>>>(gpu_x, gpu_result, x.size());
  // Copy to host and synchronise before printing (Sum result: 700000)
  float result;
  cudaMemcpy(&result, gpu_result, sizeof(float), cudaMemcpyDeviceToHost);
  cudaDeviceSynchronize();
  std::cout<<"Sum result: "<<result<<std::endl;
  // Free the memory
  cudaFree(gpu_x);
  cudaFree(gpu_result);
  return 0;
}
```

then adds `x[512]` and thread 1 `x[513]`, with the loop continuing until each thread has iterated beyond the end of the input data.

The `__syncthreads();` instruction ensures that all of the threads in the thread block have completed cumulating their part of the input data into their shared memory location before thread 0 performs the final reduction. Thread 0 does this by simply iterating over the partial results written by the other threads and cumulating them into `shared_mem[0]`, before writing the result to the global memory output address. The resulting cumulated sum can then be copied back to the CPU. As `CUDA` calls are asynchronous with respect to the CPU the `cudaDeviceSynchronize();` is used to ensure the GPU operations are complete before printing the result. Finally, it is important to note that the sum reduction kernel shown here is a relatively naive implementation and has significant room for optimisation.

Throughout the thesis many of the performance improvements are gained from simply offloading bulk, easy to parallelise computations on to the GPU. In the field of Deep Learning, much of the work in GPU performance optimisation has been abstracted in recent years and organised into higher-level Neural Network primitives in frameworks such as *Caffe* [Jia et al., 2014], *Torch* [Collobert et al., 2011], and *Tensorflow* [Abadi et al., 2015] all of which have been used in this thesis. The backends of these frameworks are primarily developed in C++/CUDA (although most also have experimental OpenCL versions for use with AMD hardware) and make use of specialised lower-level libraries such as cuDNN which have been developed by NVIDIA to speed up Deep Learning.

All Neural Network frameworks provide the most common layers already described earlier in this chapter and also provide the other required infrastructure such as data provision pipelines, optimisation algorithms, and auto-differentiation. In recent years the Python programming language has become a particularly popular language for machine learning. This is evidenced by the fact that all of the frameworks used in this thesis now provide well supported Python interfaces (unlike the others, *Tensorflow* has always been primarily designed for Python). The availability of these simpler interfaces has significantly reduced the barriers to entry for those with little GPU experience to design, train, and use Neural Networks for various tasks. Only when specialised or particularly novel operations or architectures are being explored does it become necessary for one to develop in CUDA. Even then it is only required to produce a high performance version for shorter training times. Instead slower prototypes developed in Python or C++ may be sufficient to explore an idea.

A number of high-quality open-source dense SLAM systems exist [Whelan et al., 2015a, Whelan et al., 2015b, Choi et al., 2015, Kahler et al., 2016, Dai et al., 2017b] but unlike the Deep Learning frameworks, interfacing with them frequently requires some proficiency in C++ and GPU programming. In a small step towards the low barriers to entry of Deep Learning and in many ways inspired by them, Chapter 6 of this thesis explores an object-level SLAM system that has back-end components primarily implemented in C++/CUDA which have been wrapped and incorporated into an easily accessible Python front-end.

As well as CUDA, dense SLAM systems frequently require projections of 3D data into a target camera and so they also make use of the specialised rendering operations available in the graphics pipeline of the GPU which is commonly accessed through the OpenGL API. Very usefully, OpenGL buffers can also be efficiently accessed directly from `CUDA` kernels, as will be discussed in more detail in Chapter 4. In the graphics pipeline raw 3D data such as a triangular mesh or small circular disks called surfels (used in ElasticFusion [Whelan et al., 2015a]) are rendered to a 2D image through small specialised kernels in the programmable pipeline called 'shaders.'

The first programmable shader is called a 'vertex shader' which performs per-vertex operations, the second optional shader called a 'geometry shader' takes the result as input and emits one or more primitives, and finally after these primitives are rasterised into pixel candidates (called fragments) the 'fragment shader' calculates the final depth and colour value to be written to the FrameBuffer. In ElasticFusion for example, the vertex shader transforms and projects surfel positions into the camera-view. The geometry shader then uses the surfel's radius and normal vector to emit 4 vertices as a primitive defining a projected square plane. Finally the fragment shader discards any fragment of the square which is outside of the circular radius to render the final surfel.

Although the OpenGL shader pipeline is frequently used to render colour images, it can also be re-purposed to produce images and data structures containing other useful information. For example, it is possible to render an image of unique surfel indices and to capture emitted geometry primitives directly in a TransformFeedbackBuffer rather than producing an image at all. In Chapter 3 we will describe the development of a 3D surfel map annotation tool that relies heavily on this versatility in order to allow real-time interaction with a map consisting of millions of surfels.

Chapter 3

Object Tagger

Contents

3.1 Introduction

A major challenge facing those who use CNNs for non-standard tasks is the availability of large well-annotated datasets. For certain tasks such as image classification into one of the categories of the ImageNet competition [Russakovsky et al., 2015] training data is abundant. It was the ability to make good use of this extremely large (1.2M) image training dataset in the work of [Krizhevsky et al., 2012] that initiated the current wave of research into CNNs. However when one strays off this well-trodden path, access to clean well-annotated ground truth data becomes an important challenge for practitioners. [Goodfellow et al., 2016] suggest that in practice gathering more training data is often a much better approach to improving performance than improving the learning algorithm itself.

A large task-specific dataset is very useful for improving the performance of a CNN, but it must be noted that its absence does not entirely prevent CNNs from being used. For many visual tasks, transfer learning has been found to be an effective substitute in the case of small dataset sizes [Razavian et al., 2014, Oquab et al., 2014]. Transfer learning relies on the fact that the same generic image features learned in a CNN on one task (normally image classification) continue to be very useful for a wide range of other tasks. The computational expense of training a large CNN model from scratch and the abundance of publicly available models has in fact led to fine-tuning a pre-trained model to become standard practice when dealing with smaller task-specific datasets. This has been done for a number of CNNs used in the experiments of this thesis in Chapters 4, 5, & 6.

The transferability of features in lower layers where simple edge and blob filters are learned is readily apparent. However, in higher layers features become both more co-adapted (between layers) as well as task specific [Yosinski et al., 2014]. [Yosinski et al., 2014] also show that as the required tasks become more different, the performance 'transfer gap' grows. A more thorough analysis of transferability between specific tasks has also recently been published by [Zamir et al., 2018]. In practice it therefore remains very useful to have access to a domain specific dataset. Even with pre-trained weights, access to at least a small task specific dataset is required for fine-tuning and validation, and the bigger that dataset is, the better.

One of the first per-pixel semantically annotated datasets to be released is the MSRC dataset [Shotton et al., 2006]. It consists of 591 images primarily of outdoor scenes with 21 semantic classes and a void class. Like many subsequent image segmentation datasets it is composed of images captured with a still camera. Although still useful for the present purpose, the domain of these datasets differs substantially from that of a moving camera. Not only is the temporal information required for SLAM lost but the images themselves do not contain the motion blur and variable exposure artefacts common in video trajectories. Videos also often include objects seen from less canonical viewpoints, with significant occlusions and surrounding clutter. [Brostow et al., 2009] produced the first video dataset to contain dense semantic annotations called CamVid. The labelled portion of video includes 10 minutes of outdoor driving footage with a ground truth frame sampled at 1Hz (once every 30 frames of 30Hz video) and one 6 second sequence with every other frame labelled.

Unlike image classification tasks where the ground truth label is a single category, semantic segmentation requires detailed pixel-level annotations. Of course annotation is not a pixel by pixel affair; a 100×100 image for semantic segmentation does not take 10,000 times longer to annotate than when the goal is image classification. However accurate per-pixel masks do take great effort to label. For example in the CamVid dataset it took approximately 230 man-hours to label the 701 frames using the accompanying 'InteractLabeler' tool. This tool uses automated segmentation algorithms to assist the user by defining regions which can be flood filled and allows problem areas to be refined manually. More approximate labelling tools such as LabelMe [Russell et al., 2008][1] allow frames to be annotated by drawing simple polygons around the objects. This approach is faster and annotates most of the object's visible area but it provides less precise boundary information and is still substantially more time consuming than providing a simple image classification tag.

Given the time consuming and costly nature of manual annotation, semantic segmentation datasets have historically been limited to the order of a thousand images [Shotton et al., 2006, Brostow et al., 2009, Everingham et al., 2010, Silberman et al., 2012]. Platforms such as Amazon's Mechanical Turk[2] (AMT) have in recent years assisted in scaling up segmentation datasets. The COCO dataset is an example of a much larger segmentation dataset [Lin et al., 2014] consisting of 330K labelled images, having expanded from the initial 160K release. The production of a segmentation dataset of that scale was a significant undertaking, requiring 70,000 AMT worker hours to complete [Lin et al., 2015].

In this chapter we look at some early work aimed at reducing the burden of producing well-annotated datasets for the task of indoor scene segmentation with a moving camera. It is designed with RGB-D data in mind, which is particularly useful for dense reconstruction and can be used as training input to the CNN as well as the more common RGB only inputs. We called this tool the 'Object Tagger' and it is shown in Figure 3.1. Subsequent chapters of this thesis will focus on how SLAM and Deep Learning can be usefully combined, but in this chapter we instead look at how dense SLAM can be used to assist in the generation of datasets which can be used for Deep Learning. The availability of a dense reconstructed map as well as the camera trajectory within it allows for annotations to be made on the map itself in 3D. These annotations although made only once can then be projected

[1] `https://github.com/CSAILVision/LabelMeAnnotationTool`
[2] `https://www.mturk.com/`

and overlayed per-pixel into many thousands of real RGB-D frames which would otherwise require manual frame-by-frame annotations. An additional benefit of this approach is that it also provides temporal consistency between frames which may be hard to achieve in 2D.

Even with the tools presented in this chapter, the significant resources required to produce a large-scale real-world dataset led us to quickly turn our attention to synthetic dataset generation described in Chapter 5. Therefore there remain a number of unexplored areas for future research and development in the work presented here. Much of the work aimed to provide tools to assist a user annotate a 3D map, and while in our experience we found certain tools more practical than others, it would require quantitative user studies to verify that this was the case. It also may be that some of the tools which we found less useful could be improved with further tuning, for example in the case of the 3D GrabCut tool (described in Section 3.3.3) by learning the segmentation hyperparameters. A thorough evaluation could also look at the trade-off of annotation time and accuracy of different tools.

As discussed in Section 3.4 we also found that for larger scenes the annotations occasionally became misaligned to the raw RGB-D input. We believe this could be a result of the map deformation approach in ElasticFusion, which can lead to historical poses, that are never updated, slowly becoming misaligned with the map which deforms based on later measurements. Future work could explore this problem, which may require including elements of pose-graph approaches into a dense annotation system in order to optimise the entire historic trajectory, rather than operating purely in an online manner.

3.2 Related Work

The driving inspiration behind this work was the SLAM system ElasticFusion [Whelan et al., 2015b] and the limited data availability for semantically annotated RGB-D indoor datasets. ElasticFusion provided both a readily accessible dense map and camera trajectory as well as a discretised surface representation for manual annotation. In Chapter 4 we will see a similar annotation approach refined into an automated procedure for scene understanding. The well known NYUv2 RGB-D dataset [Silberman et al., 2012] was the largest publicly available source for semantically annotated RGB-D data at the time with 1,449 annotated frames. This became one of the most

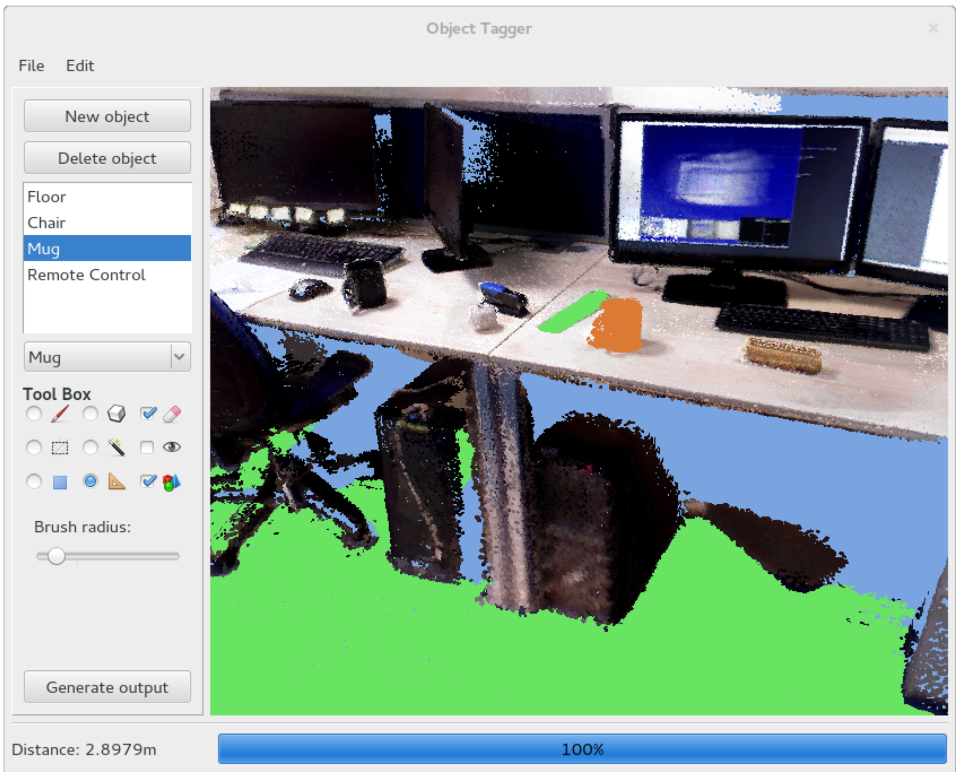


Figure 3.1: A screenshot of the final Object Tagger software being used to annotate a real scene.

used datasets within subsequent work in this thesis as it was also one of the few datasets to provide a complete video trajectory.

Some of the earliest methods to assist video semantic annotation aimed to track regions between video frames in order to propagate the 2D user annotations [Marcotegui et al., 1999, Fauqueur et al., 2007]. These 2D methods have the benefit of allowing moving objects to also be tracked between frames unlike the current system which is based on a SLAM system that assumes a static scene. However, if a static scene is assumed the process of labelling a 3D reconstruction and re-projecting it into the camera view to produce a segmentation removes the inherent difficulties of 2D region tracking between frames and instead only requires the camera pose and scene geometry to be estimated.

For the annotation of static indoor scenes a number of public datasets have also taken the approach of labelling directly in 3D. The Cornell RGB-D dataset [Koppula et al., 2011] consisted of 52 point-clouds of indoor scenes produced using RGB-D SLAM [Endres et al., 2014]. These point clouds were annotated in 3D and used

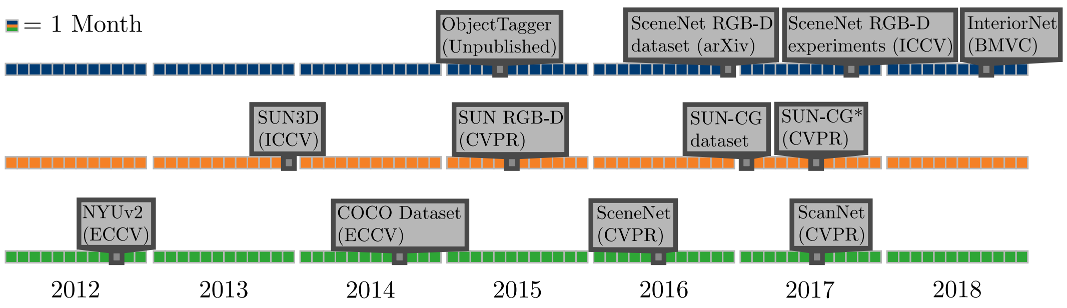


Figure 3.2: A timeline of our contributions and related work on semantic segmentation datasets. **Blue**: Our work, including the Object Tagger, the SceneNet RGB-D dataset presented in Chapter 5 (both the dataset release [McCormac et al., 2016] and later experiments [McCormac et al., 2017b]), as well the subsequent InteriorNet dataset [Li et al., 2018] on which we collaborated. **Orange**: The SUN datasets; SUN3D [Xiao et al., 2013], SUN RGB-D [Song et al., 2015] and SUN-CG [Song et al., 2017], with * denoting the SUN-CG renderings by [Zhang et al., 2017]. **Green**: Other important related datasets; NYUv2 [Silberman et al., 2012], COCO [Lin et al., 2014], SceneNet [Handa et al., 2016], and ScanNet [Dai et al., 2017a].

to validate their approach to annotating an already reconstructed 3D point-cloud. This differs from the current work which instead projects the 3D annotations back into 2D frames which have corresponding real RGB-D data to be used for validation and training data of 2D segmentation algorithms. These can be used to annotate a SLAM map in an online manner as discussed in Chapter 4.

The series of published datasets using the prefix "SUN" (short for Scene Understanding [Xiao et al., 2010]) is closely related not only to the work in this Chapter, but also to the synthetic SceneNet RGB-D dataset described in Chapter 5. To organise the ordering of contributions, Figure 3.2 provides a timeline of the most related datasets for ease of reference. The earliest related SUN publication, SUN3D [Xiao et al., 2013], produced a tool and a dataset of 8 annotated sequences which used SLAM to assist in producing 2D semantic labels. Instead of directly labelling the 3D map and projecting it, they instead take a LabelMe approach on individual frames, and use the SLAM system to propagate labels which can then later be corrected by the user if they are incorrect. They also use the labelled objects themselves as landmarks to improve the consistency of the reconstruction through bundle adjustment. This nice feature is missing from the Object Tagger, as annotation is performed on the final map alone.

The SUN RGB-D dataset [Song et al., 2015] consists of approximately 10k frames and 3D oriented bounding boxes of objects within local patches of an RGB-D point cloud. Frames of RGB-D video were used to fill holes in the depth map, but a loop-closure capable SLAM system was not used to reconstruct large areas of the scene. The 2D frames themselves were labelled using a LabelMe approach in a similar manner to the NYUv2 dataset, rather than using reprojections of more fine-grained annotated 3D surface elements as done here. They used AMT for the 2D annotations and oDesk (now upwork[3]) for the 3D annotations which took 2,051 hours to complete. Another tool designed to annotate RGB-D still frames is the SmartAnnotator tool by [Wong et al., 2015]. It assists annotation by using learned priors and the scene geometry itself to make automated structural predictions which are then refined and corrected by the user.

The present work is offline in nature using a Graphical User Interface (GUI) on a final reconstructed map. Later work by [Valentin et al., 2015] produced an online labelling tool called SemanticPaint which is both interactive and immersive. In this system the user would annotate objects in a scene by physically touching objects and speaking object class labels as voice command. A dense reconstruction would then be annotated through these touch commands assisted by a random forest and CRF backend. Our work also makes use of tools designed to assist the 3D labelling process, but these tools are limited to simple region growing and graph-cut approaches.

In more recent work direct 3D annotation has become the method of choice for many larger scale indoor scene understanding datasets. SceneNN [Hua et al., 2016] is a semantically annotated dataset of 100 indoor scene meshes which includes object poses. Initial segmentations based on [Felzenszwalb and Huttenlocher, 2004] annotations are merged using a Markov random field and fine-tuned by manual user annotations. The ScanNet dataset [Dai et al., 2017a][4] is an impressively large dataset of 2.5M images in 1.5k different scenes reconstructed using BundleFusion [Dai et al., 2017b]. They developed a WebGL interface to allow AMT workers to annotate the 3D map directly, and as here these can be projected back into the original camera viewpoint to provide pixel annotations. The map is initially automatically segmentated with [Felzenszwalb and Huttenlocher, 2004] and these regions are grouped into instances by the annotator.

Many of the datasets discussed have been generated with a 6 DoF continuous

[3]https://www.upwork.com/
[4]https://github.com/ScanNet/ScanNet

scanning trajectory of a single RGB-D camera. Another set of datasets have now been produced using the Matterport camera[5] which provides a 360° panorama of a scene at uniformly spaced viewpoints throughout a scene (often at human-height). [Armeni et al., 2017] produced a dataset of 70k images from 6 large indoor office areas, annotating the 3D point cloud directly and projecting these labels into 2D for 2D semantic annotations in a manner similar to that done here. The Matterport3D dataset [Chang et al., 2017] includes approximately 200k images from 90 buildings including private home environments. They also provide semantic and instance level segmentations of the 3D reconstructions collected by using the ScanNet interface and AMT. These annotations were then verified by specialised annotators.

Very recently the Taskonomy dataset [Zamir et al., 2018] was released, consisting of 4M images from 600 buildings. It provides RGB-D data along with numerous ground truth labels on a wide range of tasks, but unfortunately only pseudo-semantic annotations automatically generated from a supervised CNN (ResNet-151) are currently provided.

3.3 Method

The operating principle of the system is straightforward. An RGB-D video trajectory of a scene is first captured and saved. ElasticFusion processes the sequence to produce a dense surfel map and a camera trajectory. The Object Tagger then provides an interface and some tools allowing the user to annotate the map with semantic instances. Finally the annotated map is projected into the camera trajectory poses to produce pixel-level labellings corresponding to the original video sequence. As a dense reconstruction of the scene is available, occlusions of annotated objects by non-annotated geometry are properly handled without additional complications. Below we provide a brief introduction to ElasticFusion as this will also be useful for Chapter 4 and then we go on to describe the operation of the Object Tagger in more detail.

3.3.1 ElasticFusion

The Object Tagger was developed on top of the ElasticFusion SLAM system by [Whelan et al., 2015b].[6] ElasticFusion is a real-time online dense SLAM system

[5] https://matterport.com

[6] Publicly available at https://github.com/mp3guy/ElasticFusion.

capable of capturing a globally consistent surfel-based map of a static scene using an RGB-D camera. It is based on the work of [Keller et al., 2013] but includes the ability to perform loop closure. A surfel is a small disc that represents a surface patch. It can be seen rendered in the ElasticFusion GUI in the close-up view of the map in Figure 3.3. The map consists of a set $\mathcal{S}$ of independent surfels and for each surfel, index s, there is a defined position $\mathbf{p}_s \in \mathbb{R}^3$, normal $\mathbf{n}_s \in \mathbb{R}^3$, colour $\mathbf{c}_s \in \mathbb{R}^3$, radius $r_s \in \mathbb{R}$, confidence $c_s \in \mathbb{R}$, and timestamp $t_s \in \mathbb{R}$.

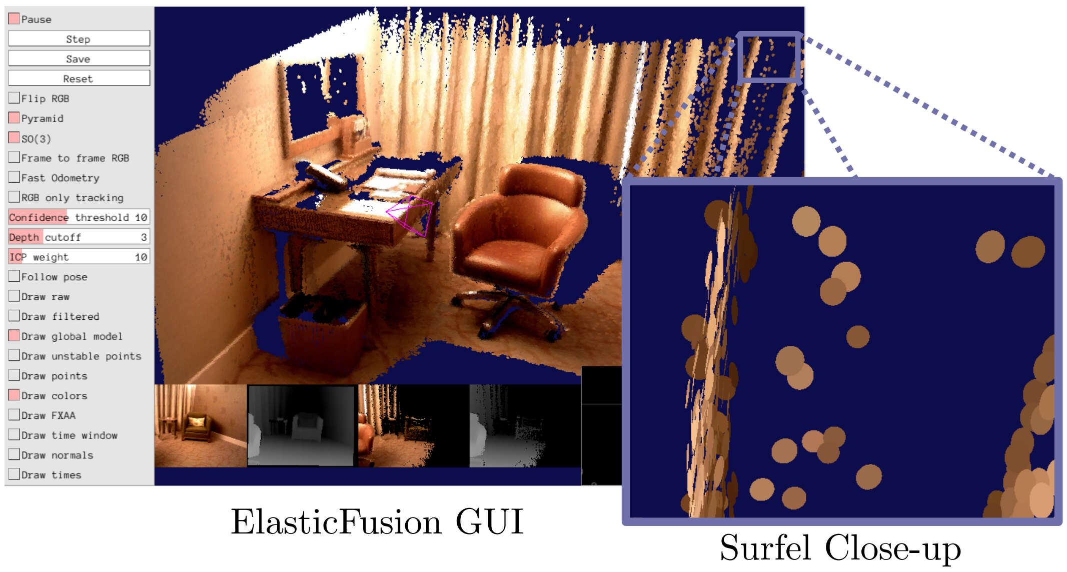


Figure 3.3: A snapshot of the ElasticFusion GUI as it processes a video sequence, in this case displaying the surfel RGB colour. The close-up on the right clearly shows the small discs from which the map reconstruction is composed.

As an RGB-D camera browses a scene, ElasticFusion alternates between mapping and tracking. Incoming depth images are denoised using a bilateral filter [Tomasi and Manduchi, 1998] and also used to estimate the normal map. Using these input measurements a new surfel is either initialised or fused with an existing point using the scheme of [Keller et al., 2013]. Measurements are associated to a single map surfel by first excluding surfels based on depth and normal difference thresholds, and then ordering the surfels by confidence and proximity to the viewing ray. A weighted average scheme is employed to fuse a measurement $\mathbf{u}_m$ associated to surfel s using a weight $W(\mathbf{u}_m)$ which is calculated as a Gaussian function based on the radial distance of the measurement pixel from the camera center. This is used to fuse the measured position $V(\mathbf{u}_m)$, normal $N(\mathbf{u}_m)$, and colour $I(\mathbf{u}_m)$ information in an identical manner:

$$\mathbf{p}_s \leftarrow \frac{\mathbf{p}_s + W(\mathbf{u}_m)V(\mathbf{u}_m)}{c_s + W(\mathbf{u}_m)}, \ \mathbf{n}_s \leftarrow \frac{\mathbf{p}_s + W(\mathbf{u}_m)N(\mathbf{u}_m)}{c_s + W(\mathbf{u}_m)}, \ \mathbf{c}_s \leftarrow \frac{\mathbf{p}_s + W(\mathbf{u}_m)I(\mathbf{u}_m)}{c_s + W(\mathbf{u}_m)}. \tag{3.1}$$

The timestamp and confidence values are also updated:

$$t_s \leftarrow t_m, \qquad c_s \leftarrow c_s + W(\mathbf{u}_m). \tag{3.2}$$

If no corresponding surfel is associated, a new one is simply initialised with the parameters that would otherwise have been fused. Surfels are initially considered 'unstable' until a confidence threshold is reached $c_s > c_{\text{thresh}}$. The radius of the surfel is calculated in order to minimise the holes in the surface $r_s = \sqrt{2}\frac{cp_z}{f}$ where f is the camera focal length [Salas-Moreno et al., 2014]. Surfels can also be removed from the map under certain conditions. A surfel that remains unstable for a given time-period is considered an outlier measurement and removed. If a stable surfel has new data merged, any occluding surfel in the model is removed as a free-space violation. Finally if a surfel has overlapping neighbour surfels with a similar position and normal information is it also removed.

In addition to the standard [Keller et al., 2013] surfel fusion scheme, the ElasticFusion map is also divided into 'active' and 'inactive' surfels. If t_s is sufficiently old compared to the current timestamp, that surfel is considered inactive and is not used for either tracking or depth fusion. This helps to prevent old areas of geometry from being disturbed when the camera pose is subject to drift before loop closure and registration. After local loop closures the matched inactive area is set to active again.

For the tracking step, ICP is used as described in Section 2.4.1 of Chapter 2. As RGB data is stored in the map, a joint cost function is used which combines both the geometric and photometric errors. For tracking, the three-channel RGB is converted to an intensity image via the weighted summation:

$$I = 0.587R + 0.114B + 0.299G. \tag{3.3}$$

The photometric residual becomes:

$$r_{\text{photo}}(\tilde{\mathbf{T}}_{WC_l}, \mathbf{u}_l) = (I_r(\mathbf{u}_r) - I_l(\mathbf{u}_l)), \tag{3.4}$$

where I_r is the rendered RGB model converted to intensity and the estimated live camera pose, $\tilde{\mathbf{T}}_{WC_l}$, is used to calculate the associated pixel in the reference image

$\mathbf{u}_r$ according to Equation 2.36. The final combined error function combines the two with a weighting w_{photo}, which is set to 0.1 by default:

$$E_{\text{total}} = E_{\text{icp}} + w_{\text{photo}} E_{\text{photo}}, \tag{3.5}$$

this error is optimised using Gauss-Newton and a three level coarse-to-fine pyramid.

A deformation graph is used to deform the map when loop closures occur. It uses a weighting technique originally designed for graphical applications [Sumner et al., 2007]. The deformation graph is computationally cheap to build and so it is rebuilt on each frame rather than modifying the existing graph. It is built by uniformly sampling the population of surfels, with connectivity based on temporal locality. This connectivity provides each surfel with a set of influencing nodes in the deformation graph. To optimise the graph, a set of constraints is formulated into a cost function which is optimised using Gauss-Newton. The cost function includes rigidity constraints, smoothness, and positional constraints.

Local loop closures can occur between the inactive and active sets of surfels using the same ICP approach already discussed. To verify this has been successful a number of criteria must be met. The residual cost must be below a specified threshold and must include a minimum number of inlier measurements, and the Hessian approximation of the system must be well conditioned. Global loop closures are detected using the real-time randomised fern encoding approach developed by [Glocker et al., 2015]. If a match is found, the same registration as for local loop closures is attempted, and the map is again deformed.

3.3.2 Annotation GUI

The GUI itself was implemented independently of ElasticFusion in Qt5.[7] New instances can be added and assigned one of the classes from a user-provided `class.cfg`, which is a plain text file listing the set of semantic classes. The annotation data stored internally for the tool is simply a single integer within each surfel denoting the annotated instance it belongs to, as well as a mapping from the instance index to the semantic class. The majority of the tool is designed to make it easy for a user to select surfels en masse in order to group them into a single object instance.

Figure 3.1 shows the appearance of the final version of the software. The toolbox and instances can be seen along the left side and the main interactive window on the

[7] `https://www.qt.io`

right allows users to navigate the space to find the best viewing angle for annotation. The currently selected instance is highlighted in orange and previously annotated objects are highlighted green. Already annotated objects can be completely hidden from view so they do not obstruct the annotation of other objects. Another viewing mode allows the user to see only the currently selected set of surfels; this viewing mode proved particularly useful for subtractive refinement of an instance after a bounding box like selection using an area-select tool. All of the tools listed below for selection can equally be used for de-selection by using the eraser checkbox.

For performance reasons the surfels are stored in an OpenGL FrameBufferObject and rendered into a Qt5 OpenGL Widget using the same surfel shaders as Elastic-Fusion. To allow users to easily select and edit large numbers of surfels the basic set of tools are the well-known paintbrush and rectangular area select tools (with example usage shown in Figure 3.4). These tools operate only on the visible surfels of the map, unlike the other tools described below. To do this, the surfels' indices are encoded and rendered to a FrameBufferObject; the paintbrush tool and rectangular select tool simply aggregate all of the surfel indices within their area of influence and assign those surfels to the current instance.

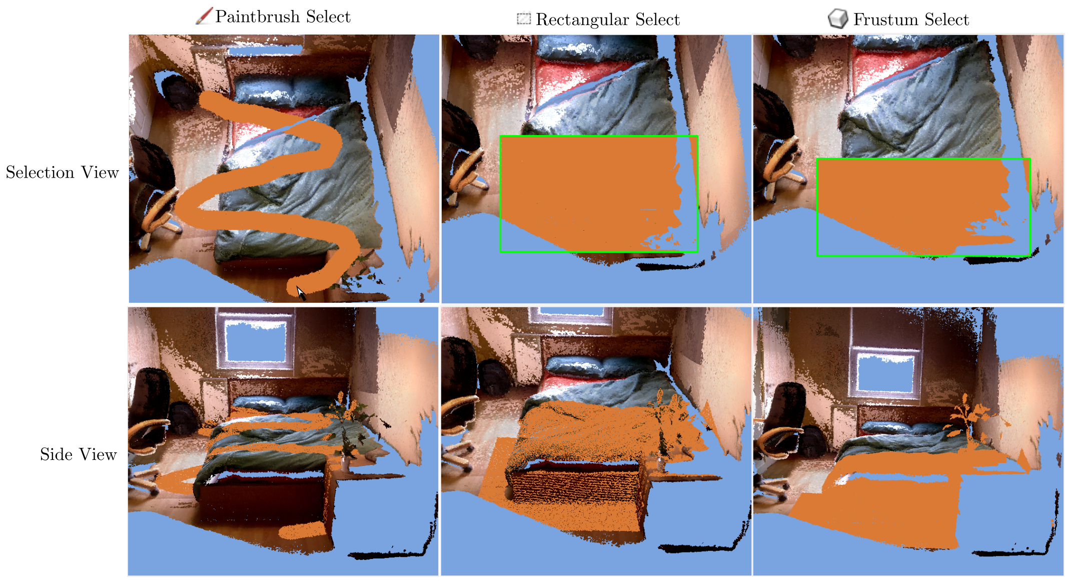


Figure 3.4: The three basic area selection tools included in the Object Tagger.

In practice the surfaces of objects are made of numerous surfels overlayed on top of each other. From one viewpoint all of the surfels may be selected but when the viewpoint changes slightly new previously occluded surfels will appear and these will not have been selected. Additionally it is sometimes helpful to select a larger volume surrounding an object so that it can be subtractively refined. For this reason the frustrum selection tool was added. A specialised vertex and geometry shader was made which accepts a user defined bounding box and filters the map for any surfel that projects into the bounding box (regardless of occlusions). These surfels' indices are written into a TransformFeedbackBuffer and assigned to the current instance.

3.3.3 3D GrabCut

The 3D GrabCut tool was created in order to simplify the process of fine-grained selection of an object from its surroundings using a rectangular selection area. It was based on the work of [Meyer and Do, 2015] who developed a 3D GrabCut tool for triangular meshes. The idea is to specify a simple 2D bounding box (corresponding to a 3D frustum) around an object of interest. The tool formulates the segmentation as a graph cut optimisation problem. Given a set of disconnected surfels we first construct the edge connections by loading the surfels into a k-d tree to allow for efficient spatial searching [Blanco and Rai, 2014].

For the set of surfels within the current view frustrum $s \in V_D$, connecting edges $(s, s') \in E_D$ are added between each surfel and its 4 nearest neighbours. The aim of the optimisation is to assign each surfel to be a member of the foreground set $\mathcal{F}$ or the background set $\mathcal{B}$. The energy function we wish to minimise by assigning particular labels to each surfel is:

$$E(S) = \sum_{s \in V_D} U(l_s) + \gamma \cdot \sum_{(s,s') \in E_D} S(l_s, l_{s'}). \tag{3.6}$$

The first term $U(l_s)$ is the data term, which defines a penalty based on the distance of a point inside the frustrum from the edge boundary, the farther the point is, especially via concave creases, the more likely that point is to be part of the foreground and the larger the penalty for assigning it a label corresponding to background. Points outside of the frustrum are given a large constant penalty K for not being labelled background:

$$U(l_s) = \begin{cases} (1 - l_s) \cdot \left(1 - e^{-G(s)}\right), & \text{if } s \in \mathcal{F} \\ l_s \cdot K, & \text{otherwise.} \end{cases} \tag{3.7}$$

Dijkstra's shortest path algorithm is used to calculate the path to the edge boundary for a given surfel denoted as $G(s)$. For two surfels s and s' with positions $\mathbf{p}$ and normals **n**, the distance between them is given by:

$$D(s', s) = \alpha \cdot \frac{d_\delta(s, s')}{\langle d_\delta \rangle} + (1 - \alpha) \cdot \frac{d_\theta(s, s')}{\langle d_\theta \rangle}, \tag{3.8}$$

with $\langle\rangle$ being used to denote the average over the selected surfels,

$$d_\delta(s', s) = ||\mathbf{p_s} - \mathbf{p_{s'}}||_2, \tag{3.9}$$

$$d_\theta(s', s) = \mu \cdot (1 - \mathbf{n_s} \cdot \mathbf{n_{s'}}), \tag{3.10}$$

were α is a balancing factor set to 0.2 and the variable μ is set to 1 for concave pairs, and 0.1 otherwise. This concavity prior is based on evidence that the human visual system defines object boundaries using concavity information [Meyer and Do, 2015]. The smoothness term simply penalises connected surfels for being assigned different labels:

$$S(s, s') = |l_s - l_{s'}| \cdot e^{-D(s,s')}. \tag{3.11}$$

The resulting graph is cut using the [Boykov and Kolomogorov, 2004] algorithm. The surfel nodes which are assigned to the foreground group are then selected. These terms and the final cut are visualised in Figure 3.5. Unfortunately the local connectivity of the surfels can lead to a relatively uniform smoothness term. This issue is exacerbated by the smoothing of normal information in the map due to the finite difference calculation.

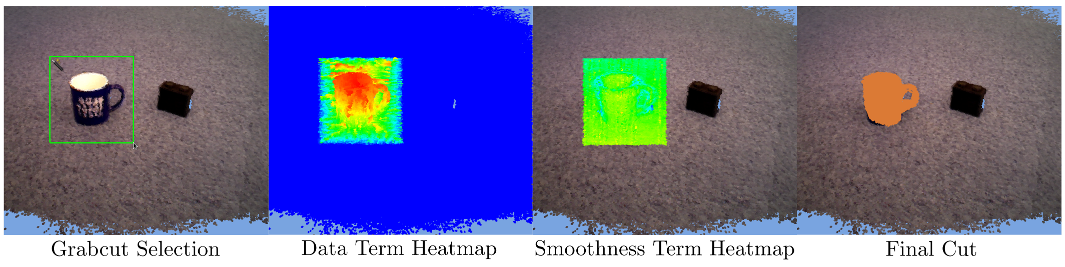


Figure 3.5: The 3D GrabCut tool operating on a simple surfel map.

3.3.4 Planar Selection

In practice we found that the 3D GrabCut tool was quite difficult to use in complex scenes. It often failed to properly segment complex geometric objects and the smoothed normals from the finite difference calculation often led to the inclusion of background geometry in the foreground segmentation. Further work could be done to improve this, for example by learning the GrabCut hyperparameters from an annotated dataset, however the ubiquity of planar supporting surfaces in indoor domestic scenes led us to to an alternative method which we found to be more effective in practice.

Instead of segmenting the complex foreground geometry in the scene first, we reversed the procedure and began by selecting the simple planar surfaces that often support objects. We found that we could robustly segment planar regions, which not only allowed the planar elements themselves to be annotated but also greatly assisted in the annotation of the objects on top of them. Once supporting surfaces are removed they often leave behind separated objects which can be easily and precisely annotated using a simple volume selection tool as illustrated in Figure 3.6.

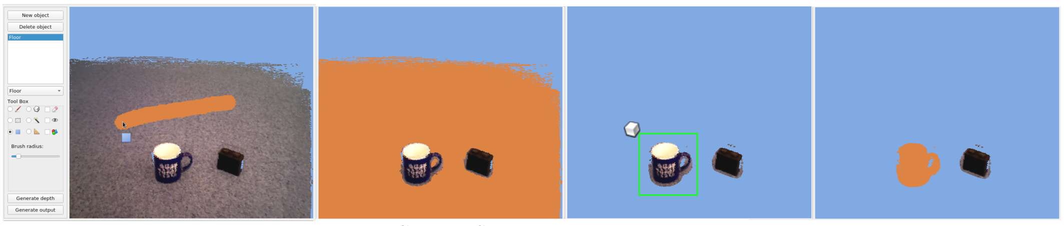


Figure 3.6: The planar select tool assists in annotating structural elements and helps separate objects of interest for annotation purposes.

The tool itself uses a very simple region growing algorithm to propagate from a set of surfels selected by the user with a paintbrush tool to an entire plane, by requiring normals of connected surfels to be within a given threshold and also by placing limits on the maximum normal difference (15°) of a connected surfel to the set selected by the user (see Algorithm 2). The connected graph used the same graphical structure as the 3D GrabCut tool described above.

3.4 Results

The final output of the tool is a video of aligned overlayed ground truth segmentation of the annotated classes which can be seen visualised in Figure 3.7. It only took fifteen minutes to annotate this scene. The work on the Object Tagger did not result in the production of a publicly available dataset, although it was used to produce a small dataset of an indoor office. This dataset is used for validation experiments in Chapter 4.

Algorithm 2: Planar Select algorithm

```
1  function Planar Select ($G, S, \theta$)
   Input: $G = (V, E)$, the connected graph of surfels and edges.
2         $S \in V$, set of selected surfels, with average normal $\mathbf{n_{avg}}$.
3         $\theta$, an angular threshold between surface normals.
   Output: $P \in V$, the final set of surfels on the plane.
4  $P = \varnothing$
5  $C = \varnothing$
6  while $S$ not empty do
7      Remove element $s$ from $S$
8      Add $s$ to $C$
9      if $\arccos(\mathbf{n_{avg}} \cdot \mathbf{n_s}) < \theta$ then
10         Add $s$ to $P$
11         foreach $s'$ connected to $s$ do
12             if $(\arccos(\mathbf{n_s} \cdot \mathbf{n_{s'}}) < \theta)$ and ($s'$ not in $C$) then
13                 Add $s'$ to $S$
14             end
15         end
16     end
17 end
18 return $P$
```

The annotations work well on relatively small scenes. However it was later found on more extensive scenes with many loop closures that the annotations, when projected into the historically estimated camera view, do not always perfectly align with the RGB-D footage. Further investigation is needed to better understand why this is the case, but it could be in part due to ElasticFusion's map-centric deformation graph approach. ElasticFusion emits a camera pose estimate at the time when the frame is first processed and that emitted pose is not updated with later measurements; instead the map geometry is deformed around the current camera pose estimate in order to close loops. When a historic camera pose is used and the final deformed map is then projected into it (having been deformed to conform to subsequent measurements), the resulting projection may no longer be well-aligned with the raw RGB-D input for the original frame.

One approach could be to refine the projected 3D annotations in 2D using the RGB-D frames themselves to better adhere to the boundaries of objects. However if the misalignment grows too large, potentially to the point of missing objects from the view entirely, a simple label refinement would be insufficient. In contrast to

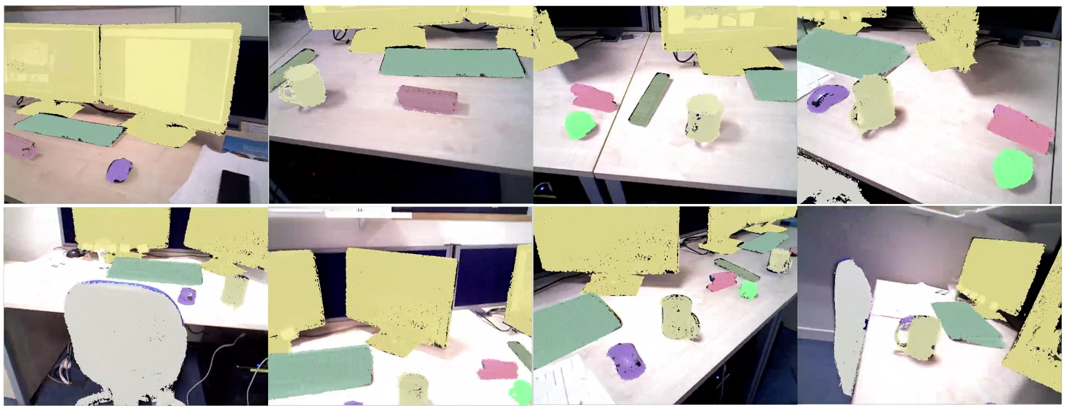


Figure 3.7: An RGB-D video sequence with the annotated labels overlayed.

the deformation graph approach, pose-graph approaches optimise the entire historic camera trajectory to best fit all of the measurements of a sequence. An alternative solution that could therefore be explored in future would be to include elements of pose-graph approaches in a dense annotation system. A more unified approach could also tighten the link between the 3D annotations, the map, and trajectory estimation itself using some of the methods explored by [Xiao et al., 2013] for the SUN3D dataset. Even when operating correctly in small scenes, it is difficult to produce a reconstruction and camera pose trajectory with enough accuracy to attain pixel-perfect reprojections. In Figure 3.7 slight misalignments are visible on the boundaries of objects, and small holes caused by missing depth data can also be seen.

3.5 Conclusion

In this chapter we highlight the practical importance of domain specific training data for Deep Learning methods. We have reviewed some of our early work which sought to reduce the burden of producing segmentation datasets with the aid of a dense SLAM system. Although we used the tool to produce some small evaluation datasets the process of scanning and annotating scenes is still slow and laborious. As discussed in Section 3.2 more recent datasets have used a similar 3D annotation approach and still required significant manual effort.

It is possible to produce huge numbers of pixel-level annotated frames from a single annotated map by simply scanning a small area for hours on end, but the value of such a dataset is quite limited. Capturing the wide variability of indoor scenes is important for CNN training in order to learn more general features applicable beyond the training set. Producing this sort of dataset still requires thousands of scans and their maps to be annotated as done, for example, in the more recent ScanNet dataset [Dai et al., 2017a].

In Chapter 5 we will return to the problem of dataset generation. In that chapter we seek to mitigate the manual effort required to capture and annotate a large-scale indoor dataset with sufficient scene variability. To do this we look to modern photorealistic rendering approaches in order to create an automated synthetic dataset generation system called SceneNet RGB-D.

Chapter 4

SemanticFusion

Contents

4.1 Introduction

An important decision that must be made in all mapping systems is that of the map representation. How can one ensure that all of the important information is included

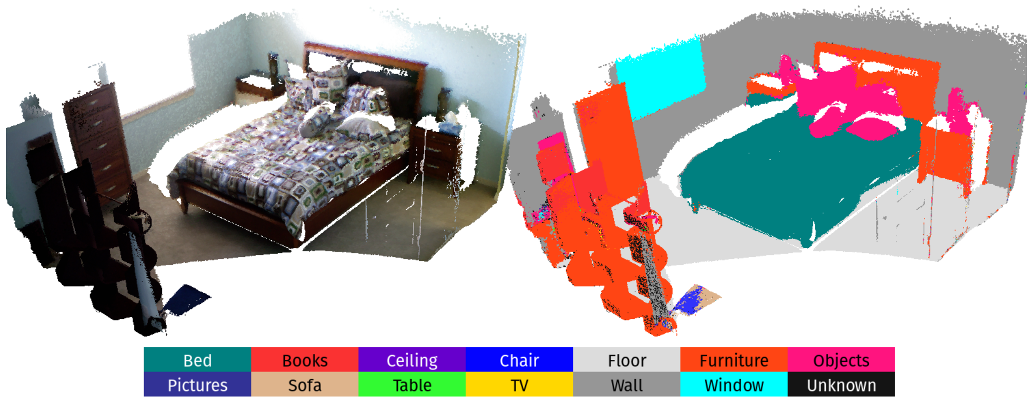
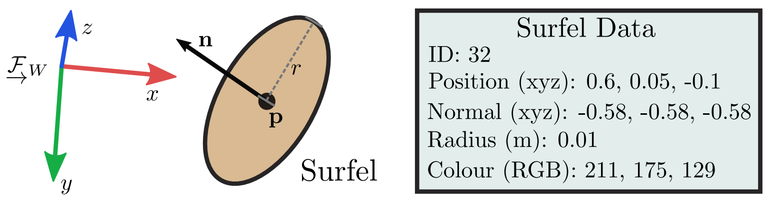


Figure 4.1: **The output of SemanticFusion**: On the left, a dense surfel based colour reconstruction of a bedroom from a video sequence in the NYUv2 test set. On the right, a semantically annotated visualisation with classes given in the legend below.

within the map? The answer depends heavily on what information is deemed to be important for the task at hand. For static scenes, accurately reconstructing the entire scene geometrically provides a great deal of information that is useful in applications ranging from robotic path planning to augmented reality.

Geometry however is not the only important attribute of a scene. Additional attributes such as texture and material properties are required to accurately visualise a scene, and can also assist in accurate tracking. In a dense geometric map it is possible to estimate and encode this information into the map's surface elements themselves. Semantic information is another important property of a scene, and in a similar manner to the material properties, a particularly straightforward manner of including semantic information within a dense map is to also densely annotate the elements within the reconstruction with semantic classes. This allows semantic information to be directly read from and visualised in the map reconstruction without fundamentally altering the underlying SLAM system.

In this chapter we combine the geometric information from a state-of-the-art dense surfel-based SLAM system ElasticFusion [Whelan et al., 2015b] which was used for generating annotated indoor scene data in the previous chapter, with advances in 2D semantic segmentation powered by CNNs. In dense SLAM systems, multiple depth readings are fused to produce accurate dense map geometry. Here multiple 2D semantic predictions produced from different viewpoints for the same map elements



Surfel Data
ID: 32
Position (xyz): 0.6, 0.05, -0.1
Normal (xyz): -0.58, -0.58, -0.58
Radius (m): 0.01
Colour (RGB): 211, 175, 129

Figure 4.2: Illustration of a surfel and an example accompanying data table.

are also fused in 3D, giving rise to the name SemanticFusion.

The core idea of the approach is to use the SLAM system as a 'correspondence engine' that can associate predictions made in 2D using a globally consistent 3D map. This allows semantic predictions from multiple viewpoints to be probabilistically fused into a dense semantically annotated map, as shown in Figure 4.1. One of the key insights is that ElasticFusion is a particularly suitable SLAM system for producing a real-time dense 3D semantic map due to its surfel-based surface representation, globally consistent map deformation approach to loop closure, and efficient runtime performance.

Surfels store geometric information such as their position, radius, and normal, but other data can also be stored in a surfel such as colour (see Figure 4.2). As a result surfels form a very natural container in which to also store semantic probability distributions. They are defined over a small surface area allowing integration of multiple readings into a reasonably sized discrete unit, while also providing spatial granularity to allow accurate boundaries in semantic segmentation.

As ElasticFusion uses a deformation graph to deform a globally consistent map for loop closures immediately, individual surfels simply carry with them their corresponding semantic information as they are deformed into the new global map shape. This greatly simplifies the process of long-term fusion of per-frame semantic predictions over wide changes in viewpoint by avoiding the complications of reintegration of semantic information required in key-frame based approaches. Instead the surfel is immediately available, with its state intact, in the new location ready for the next semantic prediction to update its state.

Most of the data for the components of this system resides on the GPU. ElasticFusion is a real-time SLAM system which uses OpenGL texture buffers to store the map. The CNN is capable of performing a prediction for a given input image in 50ms with the result being stored in a CUDA memory buffer. This allows the system as a whole to be efficiently designed for real-time use. This is done in two ways. Firstly, instead of performing a prediction for every frame, we skip a certain number of frames between predictions. The camera motion between a single frame tends to be relatively small and so the predictions are quite similar. Skipping frames means that successive updates are fusing less-correlated predictions, which in some ways assists the independence assumption made in the update scheme. Secondly, we design the map updates to occur on the GPU where the majority of the data resides to minimise device-host transfers and take advantage of the updates' highly parallel nature.

An additional benefit of projecting the 2D predictions into native 3D entities is that the geometry of the map itself provides useful information which can be used to regularise the final semantic predictions. In this chapter a straightforward Conditional Random Field (CRF) is employed which operates on the full 3D map. Although it does somewhat improve the final results, the CRF used did not lead to significant improvements. However as discussed below in Section 4.2, subsequent related work has since been developed and shown improved performance with more refined CRF schemes. Here the regularisation is a one-way system, with geometry being employed to assist in semantic annotation. However it is also entirely possible to jointly optimise both, allowing semantic information to modify and refine the geometry as well as assist with tracking. Research in this area is also discussed in Section 4.2 below.

4.2 Related Work

Two of the most closely related pieces of previous work are [Stückler et al., 2015] and [Hermans et al., 2014] which both aim towards a dense, semantically annotated 3D map of indoor scenes. They both obtain per-pixel label predictions for incoming frames using Random Decision Forests, whereas here we use a Convolutional Neural Network. They both also fuse predictions from different viewpoints in a classic Bayesian framework. [Stückler et al., 2015] used a Multi-Resolution Surfel Map-based SLAM system capable of operating at 12.8Hz. However, unlike our system, they do not maintain a single global semantic map as local key frames store aggregated semantic information, and these are subject to graph optimisation in each frame.

[Hermans et al., 2014] did not use the capability of a full SLAM system with explicit loop closure: they registered the predictions in the reference frames using camera tracking only. Their run-time performance was 4.6Hz, which would prohibit processing a live video feed, whereas our system is capable of operating online and interactively. As here, they also experiment with regularising their predictions using [Krähenbühl and Koltun, 2011]'s fully-connected CRF inference scheme to obtain a final semantic map.

Another closely related piece of work by [Cavallari and Di Stefano, 2016c] approaches the problem of dense annotation using a KinectFusion [Newcombe et al., 2011a] map. They also use a CNN to provide the semantic predictions for annotating a 3D map. An issue they encountered when adding semantics to a 512^3 resolution TSDF volume was that the naive approach of storing a class probability distribution within each voxel required 7GB for just 13 semantic classes. To circumvent this problem, they store a single identifier of the most likely class along with its score. This approach reduces the system's memory usage which no longer grows linearly with the number of classes. Updates proceed by incrementing the score of a given class by the prediction's confidence if a prediction is the same as the category stored there and decrementing it if it is different. When the score becomes negative the class can change. This work was also later extended [Cavallari and Di Stefano, 2016a, Cavallari, 2017] to work in real-time over larger scenes using the more memory-efficient TSDF encoding scheme of VoxelHashing [Nießner et al., 2013].

In our case the surfel map is natively parameterised at a surface level and so does not exhibit the cubic scaling memory requirements of a naive TSDF. A relatively large indoor map of an office may consist of $\approx$4M surfels which would mean that storing 13 4-byte floats to represent the complete probability distribution only requires an additional 200MB. In the present work we therefore use this representation and explore a Bayesian update scheme for the surfel state. In later chapters we also explore using a TSDF as part of a semantic map. However the problem of memory usage is in that case avoided by grouping elements together and sharing their semantic data rather than having a fully dense semantic annotation for each voxel.

The majority of other previous approaches to indoor semantic labelling either focus on offline batch mapping methods [Valentin et al., 2013, Koppula et al., 2011] or on single-frame 2D segmentations which do not aim to produce a semantically annotated 3D map [Everingham et al., 2010, Silberman et al., 2012, Lin et al., 2014, Song et al., 2015]. [Valentin et al., 2013] used a CRF and a per-pixel labelling from a variant of TextonBoost to reconstruct semantic maps of both indoor and outdoor scenes. This produces a globally consistent 3D map, but inference is performed on the whole mesh once instead of incrementally fusing the predictions online. [Koppula et al., 2011] also tackle the problem on a completed 3D map, forming segments of the map into nodes of a graphical model and using hand-crafted geometric and visual features as edge potentials to infer the final semantic labelling.

A core component of our system is the advance in 2D semantic segmentation powered by CNNs. CNNs have have proven capable of both state-of-the-art accuracy and efficient test-time performance. They have have exhibited these capabilities on numerous datasets and a variety of data modalities, in particular RGB [Noh et al., 2015, Long et al., 2015], Depth [Couprie et al., 2013, Handa et al., 2016] and Normals [Eigen and Fergus, 2015, Gupta et al., 2014, Gupta et al., 2015b, Gupta et al., 2015a]. In this work we build on the CNN model proposed by [Noh et al., 2015], but we modify it to take advantage of the directly available depth data. We also explore using the CNN architecture of [Eigen and Fergus, 2015] which uses surface normals as input.

Unlike object-oriented mapping, such as SLAM++ [Salas-Moreno et al., 2013] and the system presented in Chapter 6, dense semantic annotation systems aim to annotate the entire scene. Structural elements such as walls, doors, and windows which are important for describing the extent of the room are included. In this way it follows the maxim of dense reconstruction to 'map everything,' and important information can be prevented from being lost. However it must be noted that it is also possible to customise an object-oriented mapping system to include any structural elements deemed important for a given task. This approach is well exhibited by the structural planar floor prior used in SLAM++.

4.3 Method

Our SemanticFusion pipeline is composed of three separate units: a real-time SLAM system ElasticFusion, a Convolutional Neural Network, and a Bayesian update scheme, as illustrated in Figure 4.3. The role of the SLAM system is to provide

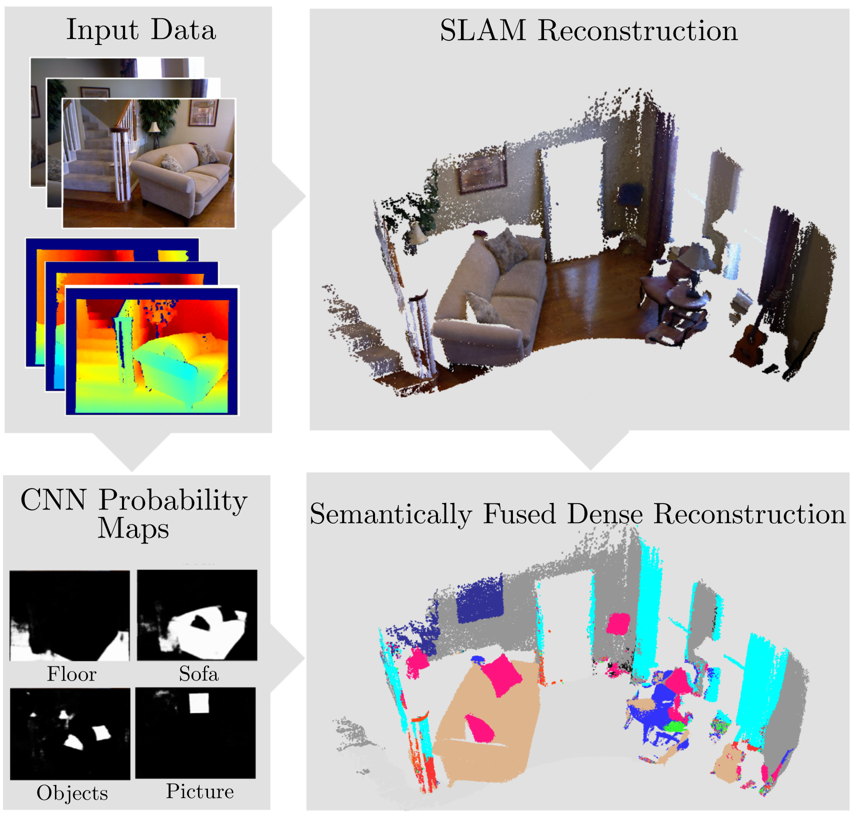


Figure 4.3: **An overview of our pipeline**: Input images are used to produce a SLAM map and a set of probability prediction maps (here only four are shown). These maps are fused into the final dense semantic map via Bayesian updates.

long-term correspondences between frames, via a globally consistent map of fused surfels. Separately, the CNN receives a 2D image of either RGB/RGB-D (for [Eigen and Fergus, 2015]'s architecture it also includes estimated normals), and returns a set of per-pixel class probabilities. A Bayesian update scheme keeps track of the class probability distribution for each surfel, and uses the correspondences provided by the SLAM system to update those probabilities based on the CNN's predictions. We also experiment with a CRF regularisation scheme to use the geometry of the map itself to improve the semantic predictions [Hermans et al., 2014, Krähenbühl and Koltun, 2011]. The following sections outline each component in more detail.

4.3.1 SLAM Mapping

As described in Chapter 3, for each arriving frame, k, ElasticFusion tracks the camera pose via a combined ICP and RGB alignment, to yield a new pose $\tilde{\mathbf{T}}_{WC}$. New surfels are added into the map using this camera pose, existing surfel information is fused with associated measurements, and inconsistent or unstable measurements are removed. Additional checks for a loop closure event run in parallel and the map is optimised immediately upon a loop closure detection. It operates at real-time frame-rates at VGA resolution and so can be used both interactively by a human or in robotic applications. We used the default parameters in the public implementation,[1] except for the depth cutoff, which we extend from 3m to 8m to allow reconstruction to occur on sequences with geometry outside of the 3m range.

4.3.2 CNN Architecture

Our CNN is implemented in Caffe [Jia et al., 2014] and adopts the Deconvolutional Semantic Segmentation network architecture proposed by [Noh et al., 2015]. Their architecture is itself based on the VGG 16-layer network [Simonyan and Zisserman, 2015], but with the addition of max unpooling and transpose convolutional layers which are trained to output a dense pixel-wise semantic probability map. This CNN was trained for RGB input, and in the following sections when using a network with this setup we describe it as 'RGB-CNN.'

Max unpooling is an approach to storing fine-grained spatial information without skip-layers in the classical sense. Instead, the information that is 'skipped' over the bottleneck are the indices of the activated pixel in the max pooling layers. Although information is still lost as all of the unactivated pixel's data is not passed on, later layers operate to 'unpool' features which have now been processed by the bottleneck to the more refined spatial location saved from lower levels. This process is illustrated in Figure 4.4.

Given the availability of depth data, we modified the original network architecture to accept depth information as a fourth channel. Unfortunately, the depth modality lacks the large scale training datasets of its RGB counterpart. The NYUv2 dataset only consists of 795 labelled training images. To use depth, we initialized the depth filters with the average intensity of the other three inputs, which had already been trained on a large dataset, and converted it from the 0–255 colour range to the 0–8m depth range by increasing the weights by a factor of $\approx 32\times$. The network diagrams for the two architectures are illustrated in Figure 4.5.

[1] Available at https://github.com/mp3guy/ElasticFusion.

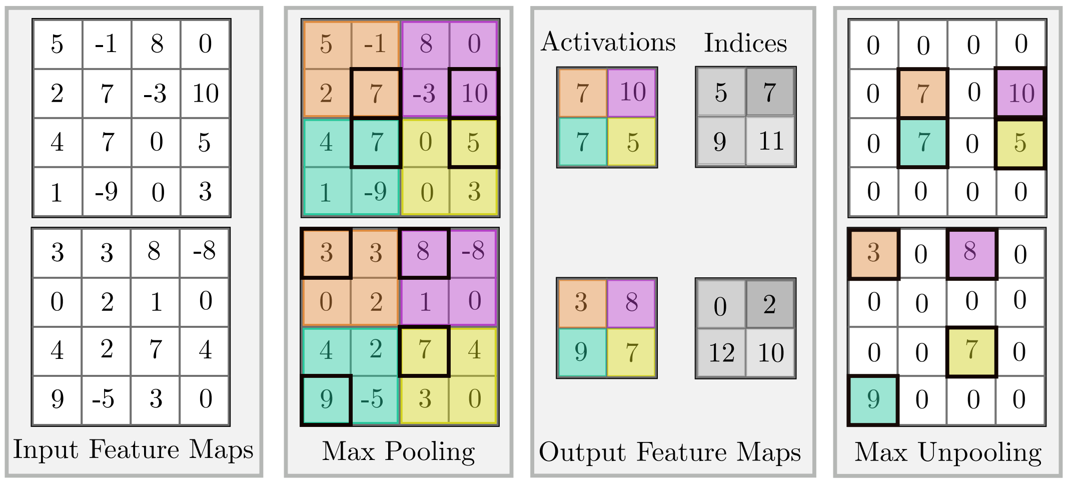


Figure 4.4: An illustration of the 'max unpooling' operation with a 2×2 kernel with a stride of 2. Here the max pooled layer is immediately unpooled for simplicity, but in practice convolutional layers operate between pooling and unpooling.

We rescale incoming images to the native 224×224 resolution for our CNNs using bilinear interpolation for RGB, and nearest neighbour for depth. As we train the RGBD-CNN on the NYUv2 training set the depth images used in training have been filled using the scheme of [Levin et al., 2004]. To maintain consistency with the base CNN in our experiments we also fill in the depth for each frame requiring depth in the same manner. However the colourisation scheme was developed in unoptimised MATLAB code. For interactive use when using the RGB-D CNN we replace this in-filling scheme with OpenCV using the inpainting approach of [Telea, 2004]. In our experiments with [Eigen and Fergus, 2015]'s implementation we rescale the inputs in the same manner to 320×240 resolution and also calculate the normals for all of the frames on which a forward pass is performed using the NYUv2 tool box [Silberman et al., 2012].[2]

4.3.3 Semantic Probability Table and Surfel Association

For each surfel (index s) in our map, $\mathcal{M}$, we store an associated discrete probability distribution, $p(c)$, over the set of all n class labels $c_i \in \mathcal{C}$. Each newly generated surfel is initialised with a uniform distribution over the semantic classes, as we begin with no *a priori* evidence as to its latent classification. Although we choose ElasticFusion as our mapping system, we keep the operation of SemanticFusion separate from the

[2]Available at `https://cs.nyu.edu/~silberman/projects/indoor_scene_seg_sup.html`.

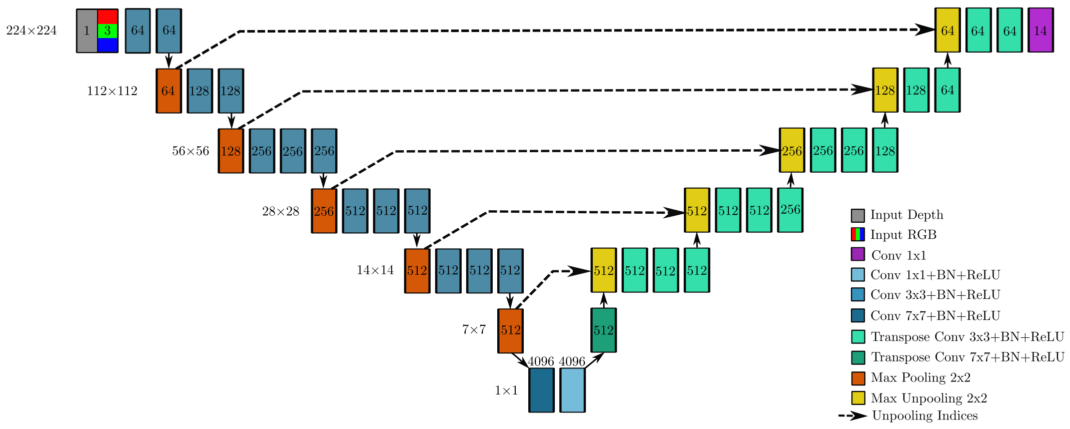


Figure 4.5: The network diagrams of the two CNN architectures used. The first set of convolutional filters are 3-channel if RGB input or 4-channel if RGBD.

map through a defined interface. Unique indices of the surfels within ElasticFusion are used to lookup and update probability values in SemanticFusion's separate table of semantic probabilities. In this way any SLAM map that can adhere to the interface and provide a globally consistent map with quantised map elements with associated indices can in principal use SemanticFusion. A simplified diagram illustrating the update procedure can be seen in Figure 4.6.

After a pre-specified number of frames, we perform a forward pass of the CNN with the image I coming from the live camera. Depending on the CNN architecture, this image can include any combination of RGB, depth, or normals. Given the data I_k of the k^{th} frame, the 224×224 output probability map P_k of the CNN provides for each pixel the probability over classes $P_k(\mathbf{u}) = p(\mathbf{u} = c_i | I_k)$.

The same frame is also processed by ElasticFusion which estimates the camera pose, $\tilde{\mathbf{T}}_{WC_k}$, and if a loop is closed the global map is deformed immediately. The map is also updated using the depth frame to either produce new surfels where required or delete surfels that were fused with other surfels or which have been unstable for a long period of time [Keller et al., 2013].

To associate surfels' positions with the corresponding CNN predictions a new OpenGL shader was added to ElasticFusion which renders the `GL_VertexID` of each surfel into an off-screen FrameBufferObject (in our case of size 640×480) using the standard pinhole camera projection given in Equation 2.18. The provision of

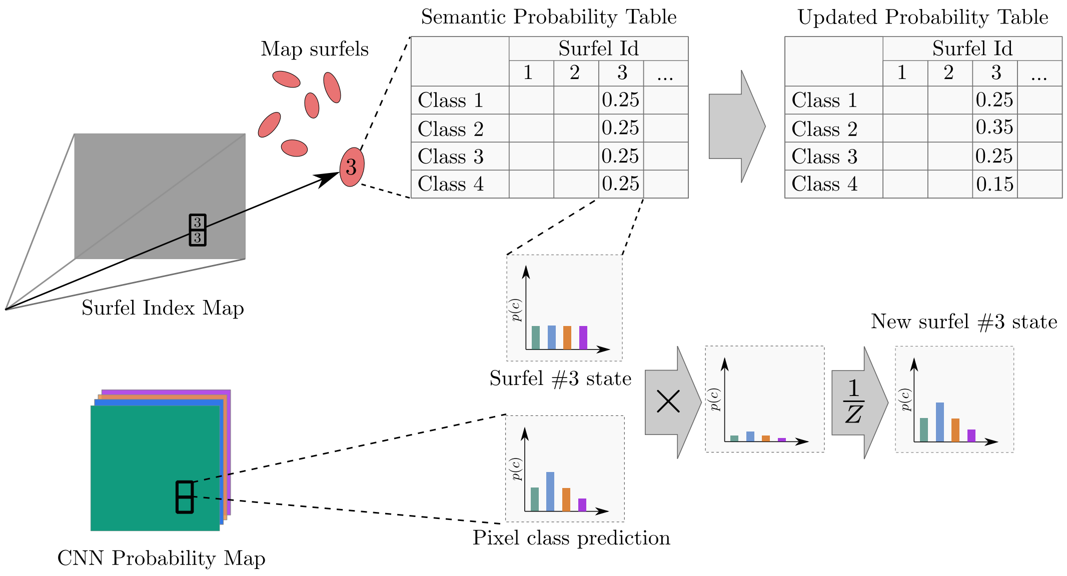


Figure 4.6: Illustration of the projective association between surfel ids and the CNN predictions, as well as the external probability table and Bayesian update scheme used by SemanticFusion.

this rendered index map is the main source of information SemanticFusion uses for its updates. The rest of the interface is to ensure the probability table stays synchronised with the maps surfels, discussed below, and to provide methods for visualisation of the semantic classes.

The surfel index map uses the `GL_VertexID` as the surfels do not explicitly store a unique identifier internally. When surfels are added or deleted the associated `GL_VertexID` in ElasticFusion's Map FrameBuffer changes. A deleted surfel results in every surfel with a higher `GL_VertexID` being 'compacted' down. For example, if the surfel with `GL_VertexID=3` is deleted, the first three surfels are unaffected but the surfel 4 will be moved to 3 and 5 to 4 and so on. New surfels are initialised from `GL_VertexID=`$|\mathcal{M}|$ onwards where $|\mathcal{M}|$ is the number of remaining surfels after the deletion step.

As illustrated in Figure 4.6, the surfel's semantic state is indexed into the probability table by its column number. As surfels are removed it is necessary to also remove them from the probability table and perform a similar compaction procedure on the table to keep the data synchronised. For this purpose the ElasticFusion map update was modified to also produce a list equal in length to the set of persistent surfels. Each entry of that list contains the previous `GL_VertexID` (before the present update) for the surfel currently in that position. The interoperability of OpenGL and CUDA is particularly useful here. It allows the OpenGL FrameBufferObject (FBO) storing this data to be registered with CUDA (using `cudaGraphicsGLRegisterBuffer`) and mapped to a CUDA accessible pointer (using `cudaGraphicsResourceGetMappedPointer`). In this way the CUDA kernels in SemanticFusion can efficiently access the memory directly in the FBO in order to synchronise the columns of the probability table.

As surfels have a spatial extent described by their normal and radius it is possible and quite likely for a single surfel's index to project on multiple pixels in the FBO. In the probability update CUDA kernel we parallelise over pixels in the index map. To prevent independent threads from editing the same surfel simultaneously each thread searches within a local patch of its own pixel location and gives priority to pixels with the same index that exist before it in scanline order. In our case, the measurements are of a lower resolution than the index map so each thread calculates its normalised image coordinates and samples the CNN probability map in a nearest neighbour fashion. If two surfels' image coordinates fall on the same pixel in the probability map, it is therefore possible they will both be updated using the same measurement probabilities.

4.3.4 Incremental Semantic Fusion

Having associated a surfel in the 3D map to both the semantic probability table and the new measurement, the surfel's state can now be updated. This update occurs by means of a recursive Bayesian update derived by [Hermans et al., 2014] and outlined below. Given time-series of measurements, $I_{0..k}$, we aim to predict the probability distribution, $p(c)$, over a set of classes $\mathcal{C}$. We are interested in obtaining the probability distribution over classes given the measurements up to that point, $p(c_k|I_{0..k})$.

Beginning with Bayes' rule:

$$p(c_k|I_{0..k}) = p(c_k|I_k, I_{0..k-1}), \tag{4.1}$$

$$= \frac{p(I_k|c_k, I_{0..k-1})p(c_k|I_{0..k-1})}{\sum_{c\in\mathcal{C}} p(I_k|c_k, I_{0..k-1})p(c_k|I_{0..k-1})}, \tag{4.2}$$

we then make the first order Markov assumption that the current measurement is conditionally independent of past measurements given $p(c_k)$:

$$p(I_k|c_k, I_{0..k-1}) = p(I_k|c_k), \tag{4.3}$$

and substituting Equation 4.3 into Equation 4.1 gives:

$$p(c_k|I_{0..k}) = \frac{p(I_k|c_k)p(c_k|I_{0..k-1})}{\sum\limits_{c\in(C)} p(I_k|c_k)p(c_k|I_{0..k-1})}. \tag{4.4}$$

Given that our beliefs over classes do not change in the absence of measurements (i.e. class distributions are not modelled to be time varying):

$$p(c_k|I_{0..k-1}) = p(c_{k-1}|I_{0..k-1}), \tag{4.5}$$

Equation 4.4 becomes:

$$p(c_k|I_{0..k}) = \frac{p(I_k|c_k)p(c_{k-1}|I_{0..t-1})}{\sum\limits_{c\in\mathcal{C}} p(I_k|c_k)p(c_t|I_{0..k-1})}. \tag{4.6}$$

As $\sum\limits_{c\in\mathcal{C}} p(c_k|I_{0..k}) = 1$ we can use a normalisation constant, Z_k, for the values which are the same for any given class, c_k:

$$p(c_k|I_{0..k}) = \frac{p(I_k|c_k)p(c_{k-1}|I_{0..k-1})}{Z_k}. \tag{4.7}$$

When running the model online we assume we get an estimate of the current class, $\tilde{p}(c_k|I_k)$. Using Bayes' rule again,

$$p(c_k|I_{0..k}) = \frac{\tilde{p}(c_k|I_k)p(I_k)}{p(c_k)} \frac{p(c_{k-1}|I_{0..k-1})}{Z_k}. \tag{4.8}$$

As $p(I_k)$ is the same for all classes, it can also be included within the normalisation constant:

$$p(c_k|I_{0..k}) = \frac{1}{Z_k} \frac{\tilde{p}(c_k|I_k)p(c_{k-1}|I_{0..k-1})}{p(c_k)}, \tag{4.9}$$

and here we assume a uniform class prior so it too can be included in Z_k which gives us the final update Equation:

$$p(c_k|I_{0..k}) = \frac{1}{Z_k}\tilde{p}(c_k|I_k)p(c_{k-1}|I_{0..k-1}), \tag{4.10}$$

where $p(c_{k-1}|I_{0..k-1})$ is the current state of a projected surfel to be updated and $\tilde{p}(c_k|I_k)$ is the new measurement associated to it as described in Section 4.3.3 above. As this procedure is independent for each surfel it can be readily parallelised in a simple CUDA kernel shown in Listing 4.1.

Listing 4.1: CUDA kernel showing a simplified example of how the surfel state update can be performed in a parallel manner on a GPU.

```
__global__
void semanticFusion(cudaTextureObject_t index_map, const uint2 index_size,
                    const float* cnn_probabilities, const uint2 cnn_size,
                    float* probability_table, const int num_classes) {
  const int x = blockIdx.x * blockDim.x + threadIdx.x;
  const int y = blockIdx.y * blockDim.y + threadIdx.y;
  // Read thread's surfel_id from texture memory and ensure a surfel exists
  const int surfel_id = tex2D<int>(index_map,x,y);
  if (surfel_id < 0) return;
  // Check for conflicting threads with same surfel_id in search window
  const int window_size = 16;
  const int h_min = max(0, y - window_size);
  const int w_min = max(0, x - window_size);
  const int w_max = min(index_size.x, x + window_size);
  // Give priority to threads in scanline order
  for (int h = h_min; h <= y; ++h) {
    for (int w = w_min; w < w_max; ++w) {
      if (h == y && w == x) {
        // No other pixel with matching surfel_id within window - update
        break;
      } else if (surfel_id == tex2D<int>(index_map,w,h)) {
        // Found same surfel_id in scanline window - do not perform update
        return;
      }
    }
  }
  // Sample measurement from CNN probability map
  const float2 norm_xy = (make_float2(x,y) + 0.5) / make_float2(index_size);
  const uint2 cnn = make_uint2((norm_xy * make_float2(cnn_size)) + 0.5);
  const int cnn_offset = num_classes * (cnn.y * cnn_size.x + cnn.x);
  const float* measurement = cnn_probabilities + cnn_offset;
  // Use the surfel_id to find the surfel probability table entry
  const int surfel_offset = (num_classes * surfel_id);
  float* surfel_state = probability_table + surfel_offset;
  // Perform the update over all classes
  float normaliser = 0.0;
  for (int class_id = 0; class_id < num_classes; ++class_id) {
    surfel_state[class_id] *= measurement[class_id];
    normaliser += surfel_state[class_id];
  }
  // Renormalise the semantic distribution
  for (int class_id = 0; class_id < num_classes; ++class_id) {
    surfel_state[class_id] /= normaliser;
  }
}
```

4.3.5 Map Regularisation

We explore the benefits of using map geometry to regularise predictions by applying a fully-connected CRF [Krähenbühl and Koltun, 2011] with Gaussian edge potentials to surfels in the 3D world frame, as in [Hermans et al., 2014].

We do not use the CRF to arrive at a final prediction for each surfel, but instead use it incrementally to update the probability distributions. In our work, we treat each surfel as a node in the graph. The algorithm uses the mean-field approximation and a message passing scheme to efficiently infer the latent variables that approximately minimise the Gibbs energy E of a labelling, $\mathbf{x}$, in a fully-connected graph, where $x_s \in \{l_i\}$ denotes a given labelling for the surfel with index s.

The energy $E(\mathbf{x})$ consists of two parts. The unary data term $\psi_u(x_s)$ is a function of a given label, and is parameterised by the internal probability distribution of the surfel from fusing multiple CNN predictions as described above. The pairwise smoothness term, $\psi_p(x_s, x_{s'})$ is a function of the labelling of two connected surfels in the graph, and is parameterised by the geometry of the map:

$$E(\mathbf{x}) = \sum_s \psi_u(x_s) + \sum_{s<s'} \psi_p(x_s, x_{s'}). \tag{4.11}$$

For the data term we simply use the negative logarithm of the chosen labelling's probability for a given surfel,

$$\psi_u(x_s) = -\log(P(L_s = x_s | \boldsymbol{I}_{1,\dots,k})). \tag{4.12}$$

In the scheme proposed by Krähenbühl and Koltun [Krähenbühl and Koltun, 2011] the smoothness term is constrained to be a linear combination of K Gaussian edge potential kernels, where $\mathbf{f}_s$ denotes some feature vector for surfel, s, and in our case $\mu(x_s, x_{s'})$ is given by the Potts model, $\mu(x_s, x_{s'}) = [x_s \neq x_{s'}]$:

$$\psi_p(x_s, x_{s'}) = \mu(x_s, x_{s'}) \left(\sum_{m=1}^{K} w_m k_m(\mathbf{f}_s, \mathbf{f}_{s'}) \right). \tag{4.13}$$

Following previous work [Hermans et al., 2014] we use two pairwise potentials; a bilateral appearance potential seeking to closely tie together surfels with both a similar position and appearance, and a spatial smoothing potential which enforces smooth predictions in areas with similar surface normals:

$$k_1(\mathbf{f}_s, \mathbf{f}_{s'}) = \exp\left(-\frac{|\mathbf{p}_s - \mathbf{p}_{s'}|^2}{2\theta_\alpha^2} - \frac{|\mathbf{c}_s - \mathbf{c}_{s'}|^2}{2\theta_\beta^2} \right), \tag{4.14}$$

$$k_2(\mathbf{f}_s, \mathbf{f}_{s'}) = \exp\left(-\frac{|\mathbf{p}_s - \mathbf{p}_{s'}|^2}{2\theta_\alpha^2} - \frac{|\mathbf{n}_s - \mathbf{n}_{s'}|^2}{2\theta_\gamma^2} \right). \tag{4.15}$$

We chose unit standard deviations of $\theta_\alpha = 0.05\text{m}$ in the spatial domain, $\theta_\beta = 20$ in the RGB colour domain, and $\theta_\gamma = 0.1$ radians in the angular domain. We did not tune these parameters for any particular dataset. We also maintained w_1 of 10 and w_2 of 3 for all experiments. These were the default settings in Krähenbühl and Koltun's public implementation [Krähenbühl and Koltun, 2011].[3]

4.3.6 Network Training

We initialise our CNNs with weights from [Noh et al., 2015] pre-trained for segmentation on the PASCAL VOC 2012 segmentation dataset [Everingham et al., 2010]. For depth input we initialise the fourth channel as described in Section 4.3.2, above. We fine-tuned this network on the training set of the NYUv2 dataset for the 13 semantic classes defined by [Couprie et al., 2013]. For optimisation we used SGD, with a learning rate of 0.01, momentum of 0.9, and weight decay of 5×10^{-4}. After 10k iterations we reduced the learning rate to 1×10^{-3}. We use a mini-batch size of 8, and trained the networks for a total of 20k iterations over 2 days on an NVIDIA GTX Titan X.

4.4 Experiments

A wide range of possible metrics can be used to evaluate a 3D semantic mapping system. Given the available NYUv2 dataset, in this evaluation we focus on re-projected 2D classification accuracy as well as run-time performance. We compare our method against a single-frame segmentation baseline using an identical CNN and find that our method improves upon the baseline 2D classification accuracy on both the NYUv2 dataset and our own dataset which was taken with a trajectory more suited to reconstruction. This suggests that the inclusion of SLAM not only provides an immediately useful semantic 3D map, but also that many state-of-the-art 2D single frame semantic segmentation approaches may be boosted in performance when combined with SLAM if video data is available. The focus of this thesis is on online semantic SLAM systems. Ideally the algorithms selected should be capable of real-time (interactive) use on current consumer hardware. Here we evaluate the run-time performance of this system on a random set of 30 NYUv2 sequences, and find the performance sufficient for interactive use operating on average at $\approx 25\text{Hz}$.

[3]Available at `http://www.philkr.net/home/densecrf`.

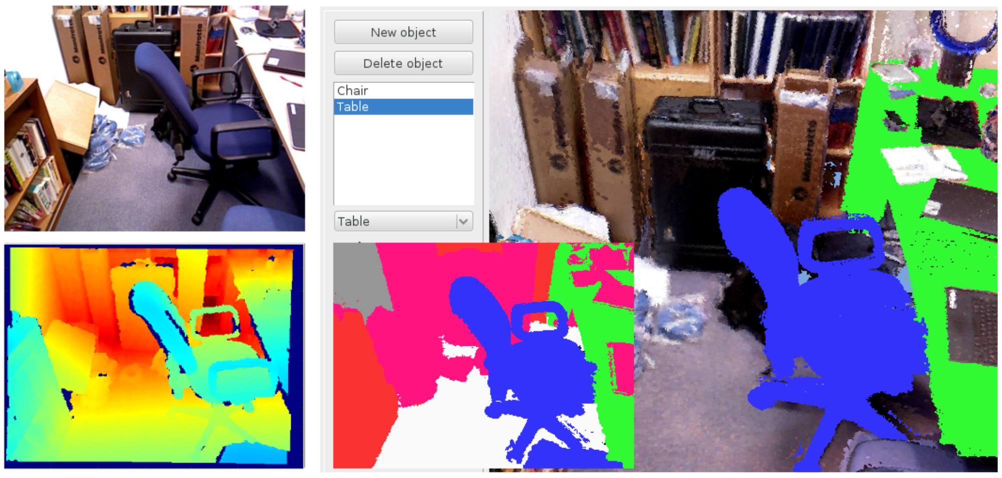


Figure 4.7: **Our office reconstruction dataset**: On the left are the captured RGB and Depth images. On the right, is our 3D reconstruction and annotation using the tool described in Chapter 3. Inset into that is the final ground truth rendered labelling we use for testing.

4.4.1 Reconstruction Dataset

Unfortunately the public NYUv2 dataset was not taken with full room reconstruction in mind, and often does not provide significant variations in viewpoints for a given scene. To explore the benefits of SemanticFusion within a more thorough reconstruction, we developed a small dataset of a reconstructed office room, annotated with the NYUv2 semantic classes using the tool developed in Chapter 3. In this dataset we aimed for a relatively complete reconstruction of an office room. The trajectory used is notably more loopy, both locally and globally, than the NYUv2 dataset which typically consists of a single back and forth sweep.

We believe the trajectory in our dataset is more representative of the scanning motion an active agent may perform when inspecting a scene. An annotated portion of the scene is shown in Figure 4.7. Every 100th frame of the sequence was used as a test sample to validate our predictions against the annotated ground truth, resulting in 49 test frames.

4.4.2 CNN and CRF Update Frequency

We used the office dataset to evaluate the accuracy of our system when only performing a CNN prediction on a subset of the incoming video frames. We used the RGB-CNN described above, and evaluated the accuracy of our system when performing a prediction on every 2^n frames, where $n \in \{0..7\}$. To illustrate the time-accuracy trade off we overlay the plot of average frame-rate for a given number of skipped frames based upon the run-time analysis discussed in Section 4.4.5. As shown in Figure 4.8, the accuracy is highest (52.5%) when every frame is processed by the network, however this leads to a significant drop in frame-rate to 8.2Hz. Processing every 10th frame results in a slightly reduced accuracy (49-51%), but over three times the frame-rate of 25.3Hz. This is the approach taken in all of our subsequent evaluations.

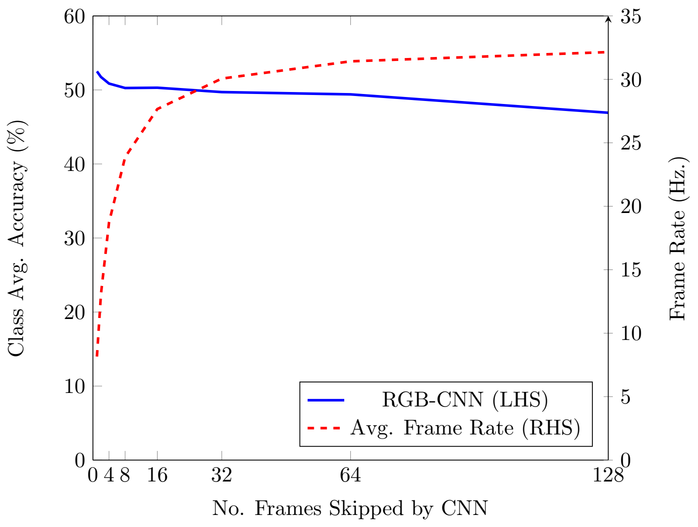


Figure 4.8: The class average accuracy of our RGB-CNN on the office reconstruction dataset against the number of frames skipped between fusing semantic predictions. We perform this evaluation without CRF smoothing. The right hand axis shows the estimated run-time performance in terms of FPS.

We also evaluated the effect of varying the number of frames between CRF updates (Figure 4.9). We found that when applied too frequently, the CRF can 'drown out' predictions of the CNN, resulting in a significant reduction in accuracy. Performing an update every 500 frames results in a very slight improvement, and so we use that as the default update rate in all subsequent experiments.

4.4.3 Accuracy Evaluation

We evaluate the accuracy of our SemanticFusion pipeline against the accuracy achieved by a single frame CNN segmentation. The results of this evaluation are summarised in Table 4.1. We observe that in all cases semantically fusing additional

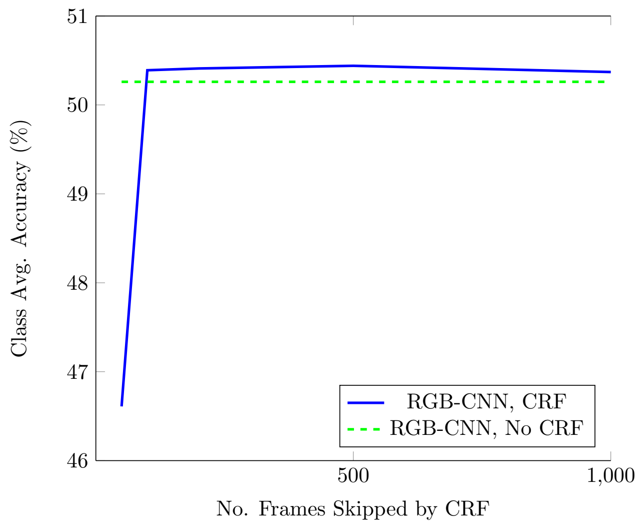


Figure 4.9: The average class accuracy processing every 10th frame with a CNN, with a variable number of frames between CRF updates. If applied too frequently the CRF was detrimental to performance, and the performance improvement from the CRF was not significant for this CNN.

viewpoints improved the accuracy of the segmentation over a single frame system. Performance improved from 43.6% for a single frame to 48.3% when projecting the predictions from the 3D SemanticFusion map.

The performance for common classes that span a large area such as ‘wall’ and ‘floor’ is notably much higher than infrequent classes that cover a smaller area and are often heavily occluded such as ‘table.’ As CNN predictions tend to be more inaccurate near the boundaries of objects, the performance on classes with proportionally more border pixels (either because of clutter or because of a smaller visible area) is generally lower. There are also fewer training examples in the dataset for these infrequent classes.

We also evaluate our system on the office dataset when using predictions from a much slower but (at the time) state-of-the-art CNN developed by [Eigen and Fergus, 2015][4] based on the VGG architecture. Although it is not possible to use the [Eigen and Fergus, 2015] CNN in real-time (it takes 1.6s to perform a prediction), we used it to evaluate whether the SemanticFusion system was capable of improving upon the predictions of even a state-of-the-art CNN, regardless of the performance

[4]We use the publicly available network weights and implementation from: http://www.cs.nyu.edu/~deigen/dnl/.

considerations. The network requires ground truth normal information, and so to ensure the input pipeline is the same as in [Eigen and Fergus, 2015], we pre-process the entire NYUv2 sequence with the MATLAB script linked to in the project page to produce the ground truth normals. With this setup we see an average improvement of 2.9% over the single frame implementation with SemanticFusion, from 57.1% to 60.0%. The performance benefit of the CRF was less clear. It provided a very small improvement of 0.5% for the [Eigen and Fergus, 2015] network, but a slight detriment to the RGBD-CNN.

In the individual class breakdown SemanticFusion produced positive results in most classes, with the notable exception of the `painting` class for the RGBD CNN. For the `painting` class the accuracy decreased from 39.5% for the single-frame prediction to 26.2% when performing SemanticFusion. A review of the sequence revealed that this was largely caused by a misclassification of pictures on the wall when the camera viewpoint was pitched up in order to reconstruct the ceiling. Although the CNN initially correctly classified most of the paintings in SemanticFusion from a level camera pose, when the camera viewpoint pitched upwards the CNN consistently misclassified pictures as either `furniture` or `objects`. This bias is likely because such an upward looking viewpoint is rarely seen in the NYUv2 training set, on which the CNN was fine-tuned. In future, this issue could potentially be mitigated by providing the CNN with geometric information about the camera's pose, either directly from the SLAM system, or via additional input channels such as surface-normal information relative to the gravity vector.

4.4.4 NYUv2 Dataset

We chose to validate our approach on the NYUv2 dataset [Silberman et al., 2012], as it was one of the few datasets which provided all of the information required to evaluate semantic RGB-D reconstruction. The SUN RGB-D [Song et al., 2015], although an order of magnitude larger than NYUv2 in terms of labelled images, did not provide the raw RGB-D videos and therefore it could not be used in our evaluation.

The NYUv2 dataset itself is still not ideally suited to the role. Many of the 206 test set video sequences exhibit significant drops in frame-rate and thus prove unsuitable for tracking and reconstruction. In our evaluations we excluded any sequence which experienced a frame-rate under 2Hz. The remaining 140 test sequences result in

Office Reconstruction: 13 Class Semantic Segmentation

Method	books	ceiling	chair	floor	objects	painting	table	wall	window	class avg.	pixel avg.
RGBD	61.8	48.2	28.6	63.9	41.8	39.5	9.1	80.6	18.9	43.6	47.0
RGBD-SF	**66.4**	**78.7**	36.8	63.4	41.9	26.2	12.1	84.2	25.3	48.3	54.7
RGBD-SF-CRF	**66.4**	78.1	37.2	64.2	40.8	27.5	10.6	85.1	22.7	48.1	54.8
[Eigen and Fergus, 2015]	57.8	54.3	57.8	72.8	49.4	77.5	24.1	81.6	38.9	57.1	62.5
Eigen-SF	60.8	58.0	62.8	74.9	**53.3**	**80.3**	**24.6**	86.3	38.8	60.0	65.8
Eigen-SF-CRF	65.9	53.3	**65.1**	**76.8**	53.1	79.6	22.0	**87.7**	**41.4**	**60.5**	**67.0**

Table 4.1: **Reconstruction dataset results**: For both our RGBD CNN and the state-of-the-art (non-real time) Eigen CNN, the SemanticFusion approach (denoted with SF) improved upon the classification accuracy of their respective baseline CNNs which perform single-frame prediction. The improvement as a result of the CRF is less significant but still generally positive. SF results were captured immediately if a keyframe was present in a strictly online fashion. When no reconstruction is present for a pixel, we fall back to the predictions of the baseline single frame network. Following previous work [Hermans et al., 2014] we exclude pixels without a corresponding depth measurement. All accuracy evaluations were performed at 320 × 240 resolution.

360 labelled test images of the original 654 image test set in NYUv2. The results of our evaluation are presented in Table 4.2 and some qualitative results are shown in Figure 4.10.

Overall, fusing semantic predictions resulted in a notable improvement over single frame predictions. However, the total relative gains of 3.3% for the RGBD-CNN was less than the 4.7% improvement witnessed in the office reconstruction dataset. We believe this is largely a result of the style of capturing NYUv2 datasets. The primarily rotational scanning pattern often used in test trajectories does not provide as many useful different viewpoints from which to fuse independent predictions. Despite this, there is still a significant accuracy improvement over the single frame predictions.

We also improved upon the state-of-the-art [Eigen and Fergus, 2015] CNN, with the class average accuracy going from 59.9% to 63.2% (+3.3%). This result clearly shows, even on this challenging dataset, the capacity of SemanticFusion to not only provide a useful semantically annotated 3D map, but also to improve the predictions of state-of-the-art 2D semantic segmentation systems.

The improvement as a result of the CRF was not particularly significant, but

NYUv2 Test Set: 13 Class Semantic Segmentation

Method	bed	books	ceiling	chair	floor	furniture	objects	painting	sofa	table	tv	wall	window	class avg.	pixel avg.
RGBD	62.5	**60.5**	35.0	51.7	92.1	54.5	**61.3**	**72.1**	34.7	26.1	32.4	86.5	53.5	55.6	62.0
RGBD-SF	61.7	58.5	43.4	58.4	92.6	63.7	59.1	66.4	47.3	34.0	33.9	86.0	60.5	58.9	67.5
RGBD-SF-CRF	62.0	58.4	43.3	59.5	**92.7**	**64.4**	58.3	65.8	**48.7**	34.3	34.3	86.3	**62.3**	59.2	67.9
[Eigen and Fergus, 2015]	42.3	49.1	73.1	72.4	85.7	60.8	46.5	57.3	38.9	42.1	68.5	85.5	55.8	59.9	66.5
Eigen-SF	47.8	50.8	79.0	73.3	90.5	62.8	46.7	64.5	45.8	**46.0**	70.7	88.5	55.2	63.2	69.3
Eigen-SF-CRF	48.3	51.5	79.0	**74.7**	90.8	63.5	46.9	63.6	46.5	45.9	**71.5**	**89.4**	55.6	**63.6**	**69.9**
[Hermans et al., 2014]	**68.4**	45.4	**83.4**	41.9	91.5	37.1	8.6	35.8	28.5	27.7	38.4	71.8	46.1	48.0	54.3

Table 4.2: **NYUv2 test set results**: Similar to the reconstruction dataset, both our RGBD CNN and the state-of-the-art (non-real time) Eigen CNN witnessed improvements upon the respective baseline single-frame classification accuracy by the inclusion of SemanticFusion (SF). However the improvement was less than that of the reconstruction dataset; this could be because of the rotational scanning pattern, which provides less viewpoint variation. The improvement as a result of the CRF is again not large but generally positive. SF results were captured immediately if a keyframe was present and when no reconstruction is available we fall back to the predictions of the baseline CNN. The accuracies of [Eigen and Fergus, 2015] were calculated from their publicly available implementation. Our results are not directly comparable with [Hermans et al., 2014] as we only evaluate on a subset of the test set, and their annotations are not available. However, we include their results for reference. Following previous work [Hermans et al., 2014] we exclude pixels without a corresponding depth measurement. All accuracy evaluations were performed at 320 × 240 resolution.

positive for both CNNs. Eigen's CNN saw +0.4% improvement, and the RGBD-CNN saw +0.3%. This could possibly be improved with proper tuning of edge potential weights and unit standard deviations, and the potential exists to explore many other kinds of map-based semantic regularisation schemes, such as those later developed by [Pham et al., 2018].

4.4.5 Run-time Performance

We benchmark the performance of our system on a random sample of 30 sequences from the NYUv2 test set. All tests were performed on an Intel Core i7-5820K 3.30GHz CPU and an NVIDIA Titan Black GPU. Our SLAM system requires 29.3ms on average to process each frame and update the map. For every frame we also update our stored surfel probability table to account for any surfels removed by the SLAM system. This process requires an additional 1.0ms. As discussed above, the other components in our system do not need to be applied for every frame. A forward pass of our CNN requires 51.2ms and our Bayesian update scheme requires a further 41.1ms. Our standard scheme performs this every 10 frames, resulting in an average frame-rate of 25.3Hz.

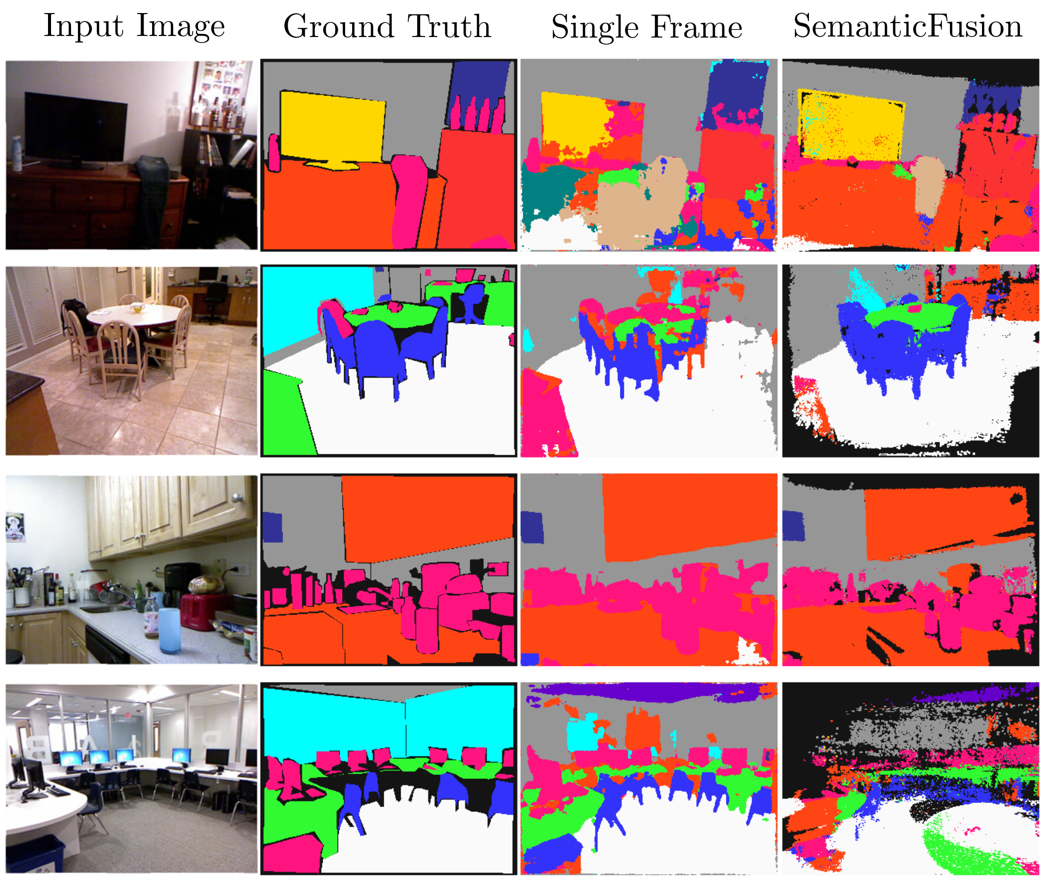


Figure 4.10: **Qualitative NYUv2 test set results**: The results of SemanticFusion shown use the RGBD-CNN with CRF after the trajectory is complete, against the same network's single frame predictions. For evaluation, the black regions of SemanticFusion denoting areas without a reconstruction are replaced with the baseline CNN predictions. Here we show only the semantic reconstruction result for clarity. The first two rows show instances where SemanticFusion has clearly improved the accuracy of the 2D annotations. The third row shows an example of a very rotational trajectory, where there is little difference as a result of fusing predictions. The final row shows an example where the trajectory was clearly not taken with reconstruction in mind, and the distant geometry leads to tracking and mapping problems even within our subset requiring 2Hz frame-rate. Cases such as this provide an advantage to the accuracy of the single frame network.

Our experimental CRF implementation was developed only for the CPU in C++, but the message passing algorithm adopted could lend itself to an optimised GPU implementation. The overhead of copying data from the GPU and performing inference on a single threaded CPU implementation is significant. Therefore on average, it takes 20.3s to perform 10 CRF iterations. In the evaluation above, we perform a CRF update once every 500 frames, but for online use it can be disabled entirely or applied once at the conclusion of a sequence.

4.5 Limitations

There are a number of limitations and areas which remain to be explored in terms of both system engineering and further research and analysis. We explored the use of two different CNNs, the CNN based on the [Noh et al., 2015] architecture, which was capable of real-time operation and the slower state-of-the-art system of [Eigen and Fergus, 2015]. However, the performance issues of the [Eigen and Fergus, 2015] CNN is quite implementation specific rather than inherent to the architecture, and so further engineering effort could allow it, or other more modern semantic segmentation systems to be used in future. We also opted for the simple approach of performing the CNN forward pass on the same thread as the main SLAM system. This simplified the system but unfortunately caused a small delay on prediction frames. In future work, the CNN forward pass and table update operations could be performed asynchronously in another thread to mitigate this issue.

The CRF regulariser used allows information from the geometry of the map to flow one way towards regularising semantic predictions. Going further, it is readily apparent, as demonstrated in a so far relatively simple manner in systems like SLAM++ [Salas-Moreno et al., 2014] that not only can reconstruction be used to provide correspondence to help labelling, but that labelling/recognition can make reconstruction and SLAM more accurate and efficient. A loop-closure capable surfel map as in ElasticFusion is highly suitable for applying operations such as class-specific smoothing (as in the extreme case of planar region recognition and fitting [Salas-Moreno et al., 2014]). The final annotated map is also highly suitable for more expensive offline post-processing which has also not been explored here. Interesting work by [Häne et al., 2013] explores joint optimisation of semantics and geometry and thereby allows the semantic predictions to directly improve the map in more complex ways than simple plane fitting and compression.

The current system is a one-way street in another way. As well as simply improving the geometry of the map, the semantic information has not been used to improve tracking or localisation. This area has been explored by [Cavallari and Di Stefano, 2016b, Cavallari, 2017] to improve upon the TSDF annotation system described in Section 4.2. Work also exists exploring using the feature maps of CNNs directly in order to improve the robustness of tracking systems [Czarnowski et al., 2017].

The recursive Bayesian update scheme used here follows the approach of [Hermans et al., 2014], which was originally developed for predictions from a Random Forest. However, research into modern CNNs suggest that while they are accurate classifiers, they also tend to be poorly calibrated and provide overconfident predictions [Guo et al., 2017]. The problem of overconfidence is further exacerbated by the independence assumption of the Bayesian update scheme, as repeated confident predictions can quickly collapse the probability distribution. These issues are not explored here but they became more apparent later when dealing with the confidence rankings of individual objects in the work in Chapter 6. Future research on this could explore the benefits of alternative fusion schemes as well as better calibration of the CNN predictions, for example by using the temperature scaling techniques proposed by [Guo et al., 2017].

There also exist opportunities for more thorough evaluation of the system as it stands. As discussed in Section 4.4.3, an upward looking camera pose seemed to have a negative impact on the accuracy of the `painting` class. Further research could more thoroughly analyse the importance of different viewing poses for certain object classes. This analysis could look not only at viewing pose relative to the global scene coordinate system, such as looking upwards, but also with respect to some canonical object pose for each class. Some of this analysis may necessitate more detailed scene annotations than available in the NYUv2, such as canonical object poses and ground-truth camera poses, as well as a greater variety of viewing poses. Some of the work described in Chapter 5 on synthetic indoor datasets may be one source of such data. With such analysis, it may be possible to improve the performance of the system by, for example, selecting prediction frames based on whether they will be particularly informative, rather than simply skipping a set number of frames between predictions. Another important factor which could be similarly explored is the effect of clutter on performance.

There is also an issue with the representation of the map itself. Although annotating a dense map certainly provides useful semantic information, interpreting and using a dense map with millions of independent semantic probability distributions is more problematic and does not provide a neat 'API' to access the map in an intuitive manner. It does not allow task-relevant queries such as 'How many chairs do we have in the conference room?' to be directly answered. As the surfels are independent it is also possible to produce quite inconsistent semantic maps. For example, if an object has different appearance characteristics on opposite sides that leads to different semantic predictions, then the map will simply label the two halves of the same object with different semantic classes, rather than recognise that these are different semantic predictions for the same entity. Many of these issues can be approached by including instance groupings within the map, an approach we explore in Chapter 6.

4.6 Conclusion

The work in this chapter confirmed our strong expectation that using a SLAM system to provide pixel-wise correspondences between frames allows the fusion of per-frame 2D segmentations into a coherent 3D semantic map. It was, to the best of our knowledge, the first time that this has been demonstrated with a real-time, loop-closure capable approach suitable for interactive room scanning. Not only that, the evaluation indicated that the incorporation of such a map led to an improvement in the corresponding 2D segmentation accuracy.

We exploited the flexibility of CNNs to improve the accuracy of a pretrained RGB network by incorporating an additional depth channel. In this work we opted for the simplest feasible solution to allow this new modality. Other work has explored other ways to incorporate depth information [Hoffman et al., 2016], but such an approach requires a duplication of the lower network parameters and was infeasible in our system due to GPU memory limitations. However we do explore architectures of this variety in the following chapter.

The reconstruction-focused office dataset showed a larger improvement in labelling accuracy via fusion than the NYUv2 dataset with less varied trajectories. This suggests viewpoint variation is an important factor for semantic fusion. It also hints at the improvements that could be achieved with significantly longer trajectories, such as those of an autonomous robot in the field making direct use of the semantically annotated 3D map.

Our experiments using [Krähenbühl and Koltun, 2011]'s fully-connected CRF inference for regularisation did not provide significant performance improvements. Subsequent work by [Pham et al., 2018] fuses a TSDF of the entire scene and semantically labels voxels using a CNN followed by a progressive CRF. Unlike here where our CRF was a slow CPU base computation, their progressive CRF operates using higher-order constraints on supervoxels in a highly efficient incremental manner enabling real-time use. That work advances beyond the Potts potential used here to include class specific relationships to form an object relationship potential, as well as including an objectness and consistency potential. They show a more significant improvement over the base SemanticFusion approach.

There are a number of problems which were encountered in this chapter which direct the research in later chapters. Given the data-hungry nature of CNNs, one problem was the lack of any large-scale RGB-D indoor datasets for training purposes. Exploration of this avenue of research had already led to the object annotation software described in Chapter 3 however producing varied data at scale using this system was still an expensive labour intensive process. To approach this problem a large scale synthetic dataset was explored and is described in the next chapter (Chapter 5). There was also the already described issue of interpreting a semantic map consisting of independent semantic probability distributions. One approach that could be taken is to cluster the already produced semantic map described here and some research is moving in this direction [Pham et al., 2018]). However, we take a different approach outlined in Chapter 6 which follows in the footsteps of SLAM++ [Salas-Moreno et al., 2013] with an instance-based SLAM system.

Chapter 5

SceneNet RGB-D

Contents

5.1 Introduction

In this chapter we explore a method to obtain and experiment with large quantities of labelled training data without the cost of manual capture and annotation. As discussed in Chapter 3, tasks which need more than a simple categorisation for an image such as semantic and instance segmentation require a pain-staking process of capturing and then annotating accurate per-pixel ground truth labels. This cost has historically limited the size of such datasets, and although the software presented in Chapter 3 mitigates some of the annotation cost for video datasets of static scenes, generating large-scale datasets would still require significant resources.

In real-world scenes it is sometimes difficult to gather some of the ground truth labels that may be desired, such as very accurate depth readings, precise camera poses, and pixel-perfect instance segmentations. These modalities often can only be estimated or potentially provided with costly additional equipment such as LIDAR for depth and VICON for camera pose tracking. In other domains such as interactive dynamic scenes, no pre-captured dataset will suffice to cover the outcome of the many possible options available to an agent. For all of the above problems, the use of synthetic data rendered using computer graphics is a potential solution. Inspired by the low cost of producing very large-scale synthetic datasets with complete and accurate ground truth information, as well as the recent successes of synthetic data for training scene understanding systems, in this chapter we aim to generate a large photorealistic dataset tailored for indoor RGB-D video trajectories.

Our dataset has several key strengths relative to previous publicly available datasets for indoor scene understanding that make it especially useful for training computer vision models which could be used for real-world applications in robotics and augmented reality. We have used ray-tracing to generate high quality synthetic RGB images, aiming towards photorealism with full lighting effects and elements such as motion blur, as well as accompanying synthetic depth images. The images are rendered from randomly generated smooth trajectories to create sequential video clips from a moving virtual camera (shown in Figure 5.1) using a novel two-body approach, which we found produced images more representative of handheld camera motion than a completely random approach.

The pipeline we have developed for generating the contents of the synthetic scenes has relied to the greatest degree possible on fully automatic methods. Object distri-

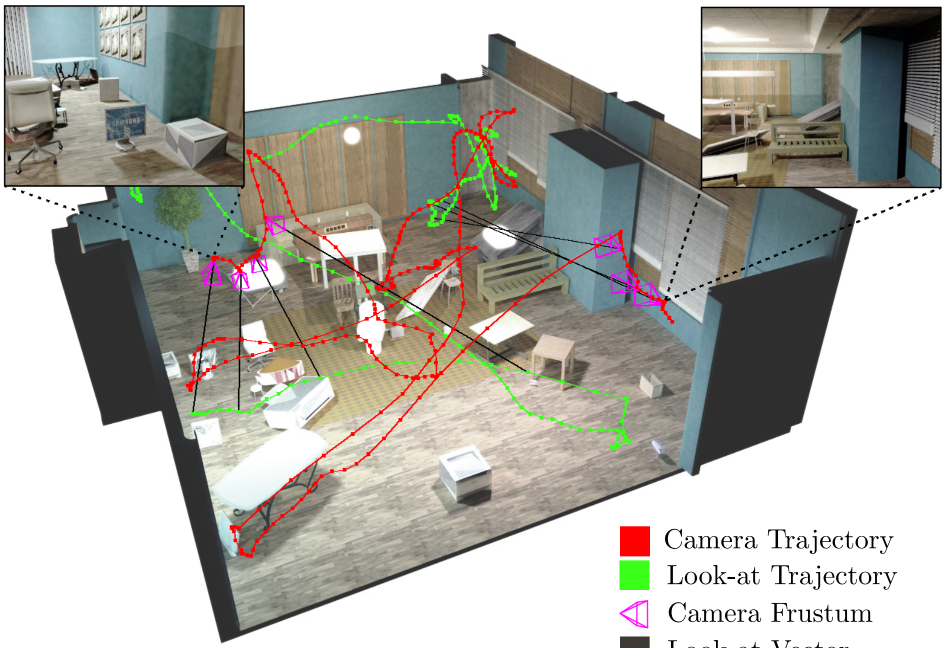


Figure 5.1: Example of a two-body camera trajectory through a synthetically generated scene with inset rendered views from the first and last camera pose.

butions are statistically sampled from publicly available real-world scene repositories and randomly positioned within a physics simulation that then ensures physically plausible scene configurations. The camera trajectory itself is also randomly generated and fully automated, while also respecting collisions in the scene geometry, unlike previous manual approaches such as inserting a captured trajectory into a scene [Handa et al., 2014]. This means that our pipeline can produce a greater degree of variety of scene configurations than others, enabling a potentially much larger dataset without the need for direct human scene design or annotation. We use this automated pipeline to generate and make available a 5M image indoor synthetic video dataset[1] of RGB-D images with accompanying ground-truth labels. We experimentally validate its usefulness for CNN pre-training when fine-tuned and tested on two real-world datasets.

[1]The dataset and open source data generation pipeline are available from the project page: https://robotvault.bitbucket.io/scenenet-rgbd.html.

5.2 Related Work

The use of synthetic data in computer vision tasks is not new. Early uses of synthetic data focused on using it for evaluation purposes because of the highly accurate ground-truth data it could provide. This approach has a long history in the field of optical flow [Barron et al., 1994, Butler et al., 2012] and has also been used for evaluation of visual odometry and SLAM systems [Handa et al., 2012, Handa et al., 2014]. A growing body of more recent research has highlighted that synthetic training data can be an effective substitute for real-world labelled data in problems where ground truth data is difficult to obtain. [Aubry et al., 2014] used synthetic 3D CAD models for learning visual elements to do 2D-3D alignment in images, and similarly [Gupta et al., 2015b] trained on renderings of synthetic objects to align 3D models with RGB-D images. [Peng et al., 2015] augmented small datasets of objects with renderings of synthetic 3D objects with random textures and backgrounds to improve object detection performance. FlowNet [Fischer et al., 2015] and FlowNet 2.0 [Ilg et al., 2017] both used training data obtained from synthetic flying chairs for optical flow estimation; and [DeSouza et al., 2017] used procedural generation of human actions with computer graphics to generate a large dataset of videos for human action recognition.

For outdoor scenes, [Ros et al., 2016] generated the SYNTHIA dataset for road scene understanding, and two independent pieces of work by [Richter et al., 2016] and [Shafaei et al., 2016] produced synthetic training data from photorealistic gaming engines, validating the performance on real-world segmentation tasks. [Gaidon et al., 2016] used the Unity engine to create the Virtual KITTI dataset, which takes real-world seed videos to produce photorealistic synthetic variations to evaluate robustness of models to various visual factors. For indoor scenes, recent work by [Qiu and Yuille, 2016] called UnrealCV provided a plugin to generate ground truth data and photorealistic images from the UnrealEngine. This use of gaming engines is an exciting direction, but it can be limited by proprietary issues either by the engine or the assets. Our SceneNet RGB-D dataset uses open-source scene layouts and objects and for rendering we have built upon the ray-tracing framework OptiX[2] which allows significant flexibility in the ground truth data we can collect and visual effects we can simulate.

[2] https://developer.nvidia.com/optix

The inspiration for the present work was the SceneNet dataset produced by [Handa et al., 2016]. SceneNet consisted of a repository of labelled synthetic 3D scenes from five different categories. That repository was used to generate per-pixel semantic segmentation ground truth for depth-only images from random viewpoints. They demonstrated that a CNN trained on 10k images of synthetic depth data and fine-tuned on the original NYUv2 [Silberman et al., 2012] and SUN RGB-D [Song et al., 2015] real image datasets showed an increase in the performance of semantic segmentation when compared to a network trained on just the original datasets.

Concurrent work by [Song et al., 2017] produced the SUN-CG dataset which contains ≈46k synthetic scene layouts created using Planner5D.[3] The most closely related approach to ours is the follow-up work of [Zhang et al., 2017] which used the SUN-CG layouts to generate 400k physically-based RGB renderings of a randomly sampled still camera within those indoor scenes. They also provided ground truth labels for three selected tasks: normal estimation, semantic annotation, and object boundary prediction. They compared pre-training a CNN (already with ImageNet initialisation) on lower quality OpenGL renderings against pre-training on high quality physically-based renderings, and found pre-training on high quality renderings outperformed on all three tasks.

Our dataset, SceneNet RGB-D, samples random layouts from SceneNet [Handa et al., 2016] and objects from ShapeNet [Chang et al., 2015] to create a practically unlimited number of scene configurations. As shown in Table 5.1, there are a number of key differences between our work and others that were available at the time. Firstly, our dataset explicitly provides a randomly generated sequential video trajectory within a scene, allowing 3D correspondences between viewpoints for 3D scene understanding tasks, with the ground truth camera poses acting in lieu of a SLAM system. Secondly, [Zhang et al., 2017] use manually designed scenes, while our randomised approach produces chaotic configurations that can be generated on-the-fly with little chance of repeating. Moreover, the layout textures, lighting, and camera trajectories are all randomised, allowing us to generate a wide variety of geometrically identical but visually differing renders as shown in Figure 5.10.

We believe such randomness could help prevent overfitting by providing large quantities of less predictable training examples with high instructional value. Addi-

[3] https://planner5d.com/

	NYUv2 [Silberman et al., 2012]	SUN RGB-D [Song et al., 2015]	sceneNN [Hua et al., 2016]	2D-3D-S [Armeni et al., 2017]	ScanNet [Dai et al., 2017a]	SceneNet [Handa et al., 2016]	SUN CG* [Song et al., 2017]*	**SceneNet RGB-D** [McCormac et al., 2017b]
RGB-D videos available	✓	✗	✗	✓	✓	✗	✗	✓
Per-pixel annotations	Key frames	Key frames	Videos	Videos	Videos	Key frames	Key Frames	Videos
Trajectory ground truth	✗	✗	✗	✓	✓	✗	✓	✓
RGB texturing	Real	Real	Real	Real	Real	Depth	Photorealistic	Photorealistic
Number of layouts	464	-	100	270	1513	57	45,622	57
Number of configurations	464	-	100	270	1513	1000	45,622	16,895
Number of annotated frames	1,449	10k	-	70k	2.5M	10k	400k	5M
3D models available	✗	✗	✓	✓	✓	✓	✓	✓
Method of design	Real	Real	Real	Real	Real	Random & Manual	Manual	Random

Table 5.1: A comparison table of 3D indoor scene datasets and their differing characteristics at the time of SceneNet RGB-D's publication. SceneNN provides annotated 3D meshes instead of frames. 2D-3D-S provides rotational scans at positions rather than free-moving 3D trajectories. *We combine within this column the additional [Zhang et al., 2017] work of physically-based renderings of the same scenes.

tionally, randomness provides a simple baseline approach against which more complex scene-grammars can justify their added complexity. It remains an open question whether randomness is preferable to designed scenes for learning algorithms. Randomness leads to a simpler data generation pipeline and, given a sufficient computational budget, allows for dynamic on-the-fly generated training examples suitable for active machine learning. A combination of the two approaches, with reasonable manually designed scene layouts or semantic constraints alongside physically simulated randomness may provide the best of both worlds, and is just recently starting to be explored [Li et al., 2018].

5.3 Dataset Overview

Our pipeline begins with 3D models of empty scene layouts and object models from the ShapeNet repository [Chang et al., 2015]. We sample a number of objects based on the area of the layout and a randomly selected object density. The categories of objects selected are based on the scene layout type (*e.g.* living room) so as to make semantically plausible scenes. As the ShapeNet models lack absolute scale information we first estimate a metric scale for each object and then position it randomly within the layout. Physically plausible scene configurations are created using a physics engine to simulate the scene dynamics with collisions and gravity. Within each of these generated scenes a collision-aware camera trajectory is generated to mimic human hand-held camera motion. Finally both the scene geometry and the trajectory are passed to the rendering engine which adds lighting effects and random textures to render the RGB images, depth images, and instance segmentations.

For the dataset, we had to balance the competing requirements of frame-rates for video sequences with the computational cost of rendering many very similar images, which would not provide significant variation in the training set for CNNs. We decided upon 5 minute trajectories at 320×240 image resolution, but with a single frame per second, resulting in 300 images per trajectory. Our trajectory is calculated at the full 25Hz, but we only render every 25th pose. Each pose in fact consists of a pair of poses, which define the shutter open and shutter close of the camera. We sample from poses linearly interpolated between the two points to produce motion blur artefacts.

Different ground truth labels can be obtained with an extra rendering pass. Instance labels are obtained by assigning indices to each object and rendering for each pixel the index of that object instead of RGB values. Depth is defined as the Euclidean ray length from the camera origin to the first surface it intersects. This provides perfect depth information even in the case of reflections. Our depth definition is different from that of the Kinect which gives the z-axis component of the distance along the camera's principal axis. For depth and instance indices we do not supersample each pixel as we do for RGB; instead a single ray is emitted from the centre of the pixel.

From the above explicit ground truth information and accompanying metadata it is possible to calculate a number of other pieces of implicitly defined ground truth information. For example, for each trajectory we store a mapping from instance labels to a semantic label to produce semantic segmentation data. These semantic labels are defined with a WordNet index [Princeton University, 2010], which provides a useful network structure for semantic links and hierarchies. It is also possible to calculate the instantaneous optical flow using the static scene assumption, the camera trajectory, and the depth map. Some examples of the available ground truth information for a given frame is shown in Figure 5.2.

By using the camera pose and backprojecting the depth map, it is also possible to calculate the 3D position of each surface point in $\underrightarrow{\mathcal{F}}_{W}$. We use this to calculate voxel correspondence indices (for some arbitrarily selected voxel size) for an entire trajectory to mimic the correspondences available in a perfect SLAM system using a voxel grid. Figure 5.3 shows an example colourisation of this correspondence system.

Our dataset is separated into train, validation, and test sets. Each of these sets has a unique set of layouts, objects, and trajectories particular to the set. However

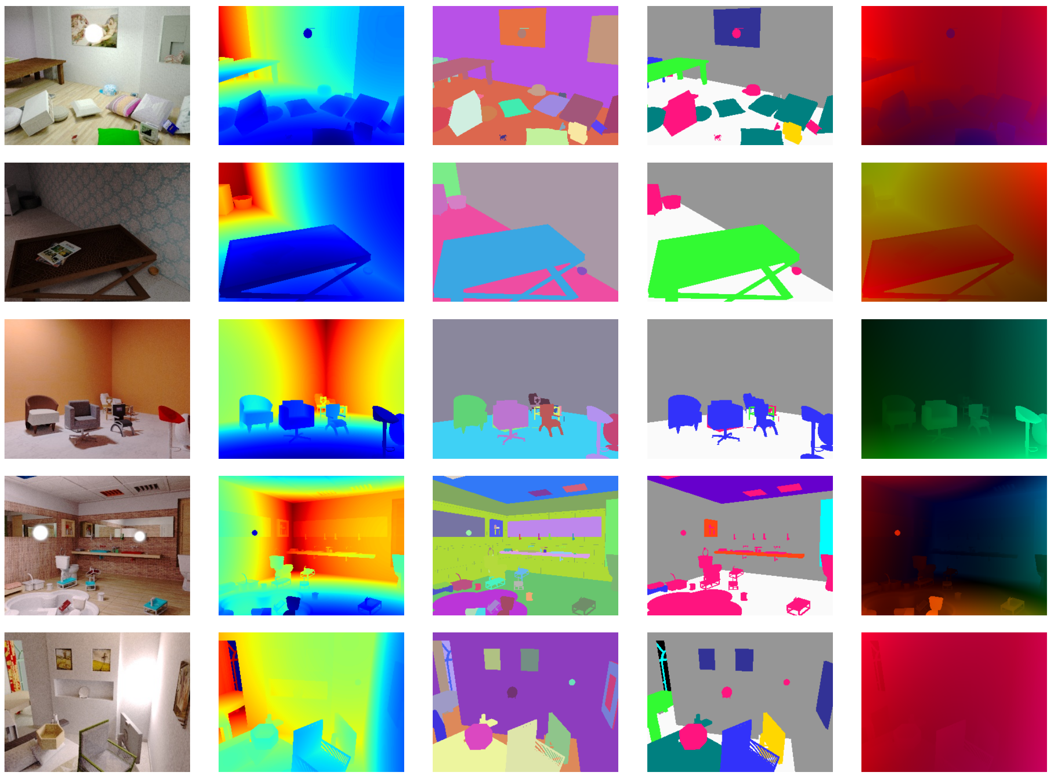


Figure 5.2: Hand-picked examples of scenes from our dataset. Rendered images on the left and the corresponding available ground truth information on the right.

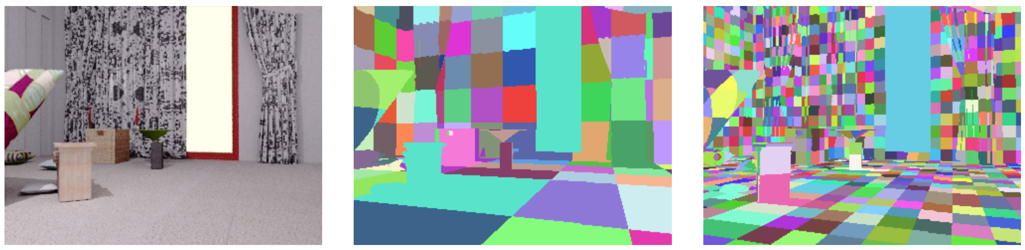


Figure 5.3: On the left is the rendered image; on the right are unique randomly coloured voxels that remain the same throughout a trajectory. Outside the window there is no depth reading so we assign all of these areas the same default identifier.

the parameters for randomly choosing lighting and trajectories remains the same. We selected two layouts from each type (bathroom, kitchen, office, living room, and bedroom) for the validation and test sets making the layout split 37-10-10. For ShapeNet objects within a scene we randomly divide the objects within each WordNet class into 80-10-10% splits for train-val-test. This ensures that some of each type of object are in each data split. Our final training set has 5M images from 16k layouts and our validation and test set have 300k images from 1k different room layouts. Each layout has a single accompanying trajectory through it.

5.4 Scene Generation

To create scenes, we first randomly choose the density of objects per square meter of layout floor area. In our case we have two of these densities. For large objects we randomly sample a density uniformly in the range $0.1 - 0.5\,\text{objects/m}^2$, and for small objects (<0.4m in height) we sample from a density in the range $0.5 - 3.0\,\text{objects/m}^2$. Given the floor area of a scene, we calculate the number objects required.

We sample objects for a given scene according to the distribution of object categories in that scene-type in the SUN-RGBD dataset [Song et al., 2015]. We do this with the aim of including semantically relevant objects within a scene context. For example a bathroom is more likely to contain a sink or toilet than a microwave (see Figure 5.4 for an object breakdown by scene type). For each sampled object type we pick a random instance uniformly from the available ShapeNet models for that object type. This approach requires accurate associations between the objects in SUN-RGBD and ShapeNet. In Figure 5.4, the unfortunate number of mailboxes in a number of layouts is a result of a mistaken mapping of the 'box' class in SUN RGB-D to a class named 'box' in ShapeNets which contains primarily mailboxes. This mishap serves to highlight some of the difficulties inherent in working with large-scale objects repositories in an automated way.

5.4.1 Estimating Metric Scales

The majority of 3D models in free 3D asset repositories are created by 3D designers and artists without any explicit designation of metric-scale information. This is a problem as it is desirable that the objects placed in our synthetic scenes have physical dimensions similar to their real world counterparts. Fortunately, datasets like SUN RGB-D [Song et al., 2015] are captured with a depth camera and provide metric 3D bounding boxes for each labelled object in the scene. We leverage this information to obtain the height distribution of object categories, and then randomly sample metric heights from this distribution to scale each object before placing it in the scene. We maintain the aspect ratio of these objects during this scaling procedure.

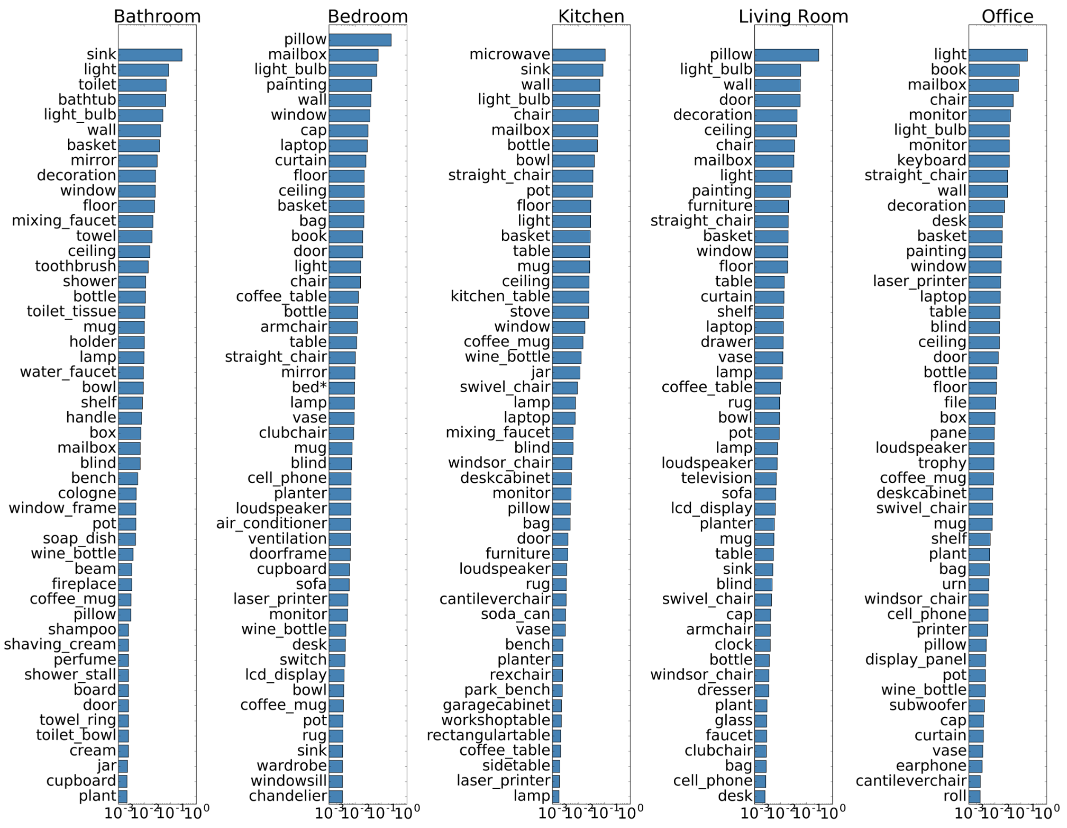
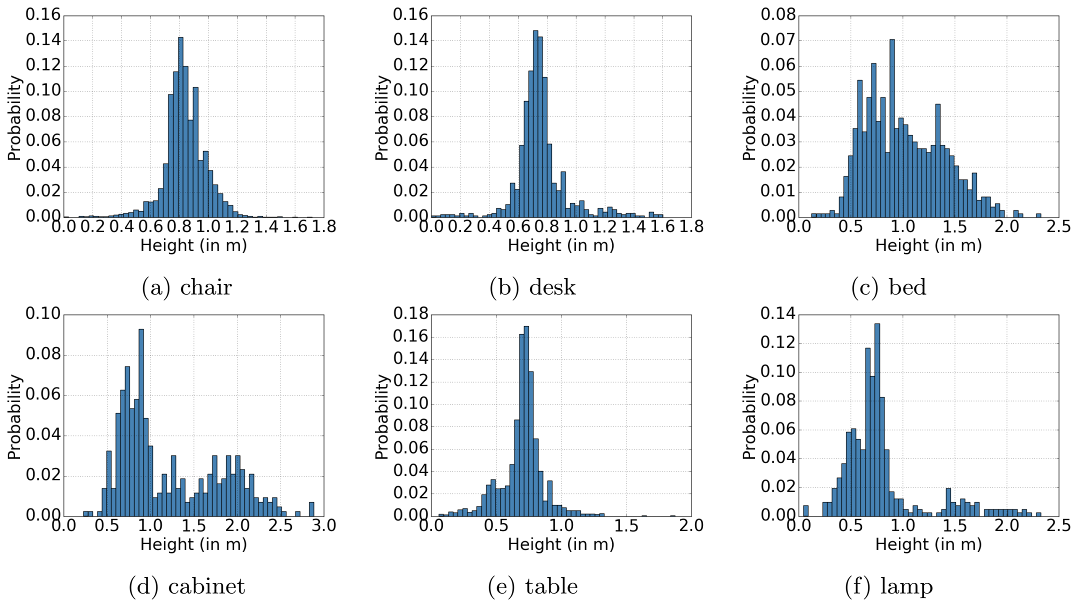


Figure 5.4: Top 50 objects and their log proportions by scene type. Note that beds are subdivided into a number of similar classes such as `miscbeds` and `kingsized beds` which we here combine these into a single group.

Figure 5.5 shows probability distributions of heights of some objects as obtained from our analysis of the SUN RGB-D dataset.

This simple approach has a number of drawbacks. The lack of granularity within classes can lead to multimodal height distributions. For example bedside lamps and floor lamps both are within the same 'lamp' class, however their heights vary significantly. If the height of a floor lamp is applied to a squat bedside lamp, the resulting object can appear closer in its dimensions to a refrigerator than any lamp. Tackling this issue is a significant problem and could be addressed in future work by using either more refined scale estimation methods [Savva et al., 2014] or a large object repository with accurately defined metric scales [Li et al., 2018].



(a) chair (b) desk (c) bed (d) cabinet (e) table (f) lamp

Figure 5.5: Probability distributions of heights of different objects as obtained from SUN RGB-D. It is interesting to see that some objects like cabinets and lamps clearly do have multimodal height distributions.

5.4.2 Generating Random Scenes with Physics

We use an off-the-shelf physics engine called Project Chrono[4] to simulate the scene dynamics to attain a final physically plausible static scene configuration. We opted for this rather than a computationally more efficient static geometric analysis method for a number of reasons. Firstly, the computational bottleneck in our system was the rendering pipeline on the GPU and the physics engine uses only the CPU. We found we could physically simulate many scenes in the same time it takes to render one trajectory, so in a sense the extra computation is 'free' for us. Secondly, the off-the-shelf physics software was readily available and quite easy to use, and early prototypes showed it resulted in qualitatively sensible layouts. Finally, although not explored here, a full physics simulator leaves open the potential for physically simulated dynamic scenes in future work.

The objects are provided with a constant mass (10kg) and a convex collision hull and positioned uniformly within the 3D space of the layouts axis-aligned bounding box. To slightly bias objects towards maintaining a correctly oriented upwards direction, we offset the center of gravity on the objects to be below the mesh.

[4] `https://projectchrono.org/`

Without this, we found that very few objects such as chairs were in their normal upright position after the physics simulation had completed. One drawback of the simplified convex collision hull is that certain objects such as tables can sometimes be propped up by a small object underneath the middle of it without any contacting model geometry.

The physics engine models the movement of objects using Newtonian laws, and their interactions with each other and the layout itself (which is properly modelled as a static non-convex collision object). We simulate 60s of the system, which we found was sufficient time to allow the objects to settle to a stable and physically realistic configuration most of the time as visualised in Figure 5.6. It is important to note that the scene is not necessarily organised and structured in a human manner. It contains objects in random poses and locations but the overall configuration is physically plausible *i.e.* we will not have configurations where an object cannot physically support another, and unrealistic object intersections are also avoided.

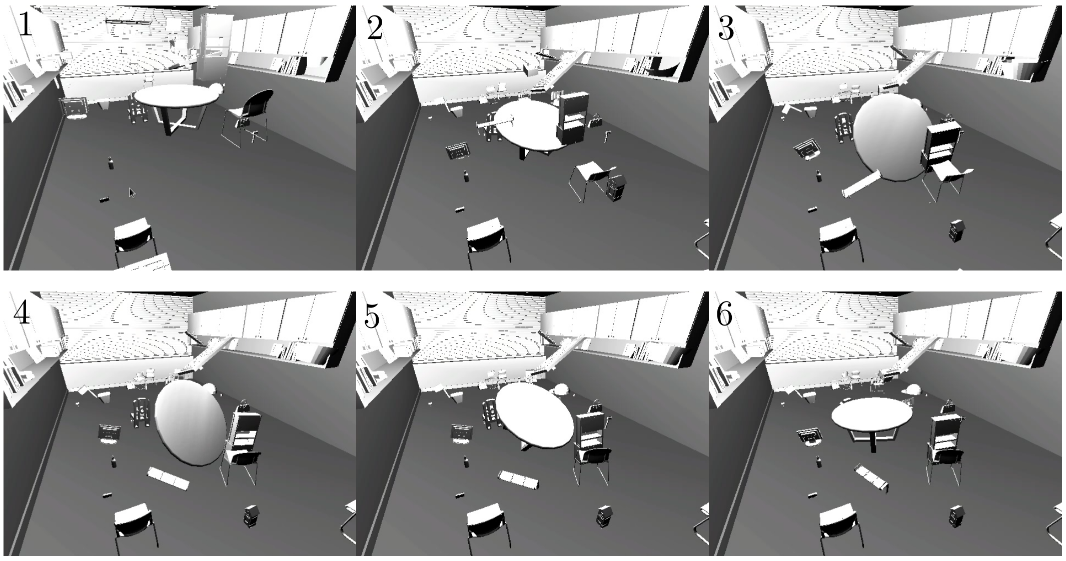


Figure 5.6: Selected still frames of a physical simulation for a randomly initialised scene. We use a physics engine to simulate 60s of scene dynamics to allow the objects to settle to a stable and physically realistic configuration.

5.5 Random Trajectory Generation

As we aim to render videos at a large scale, it is imperative that the trajectory generation be automated to avoid costly manual labour. The majority of previous approaches have used a SLAM system operated by a human to collect realistic hand-held motion. The trajectory of the camera poses returned by the SLAM system is then inserted in a synthetic scene and the corresponding data is rendered at discrete or interpolated poses of the trajectory [Handa et al., 2012, Handa et al., 2014]. Such reliance on humans to collect trajectories quickly limits the potential scale of the dataset and ignores collisions with scene geometry.

We automate this process using a simple random camera trajectory generation procedure which we have not found in any previous synthetic dataset work. For our trajectories, we have the following desiderata. Our generated trajectories should be random, but slightly biased towards looking into central areas of interest, rather than, for example panning along a wall. It should contain a mix of fast and slow rotations such as those of a human operator focusing on nearby and far away points. It should also have limited rotational freedom with emphasis on yaw and pitch rather than roll, which is a less prominent motion in hand-held or head-mounted display trajectories or ground robot trajectories.

5.5.1 Two-Body Camera Trajectories

To achieve the desired trajectory paths we simulate a random trajectory of two physical bodies in space. One defines the location of the camera and the other its 'look-at' point which serves as a proxy for a human's focus within a scene. We take the simple approach of locking roll entirely, by setting the up-vector to always be along the y-axis. With this constraint these two points then completely define the camera coordinate system. An alternative more realistic approach could in future allowing slight deviations from the y-axis.

A physically-based approach to simulating the two bodies has a number of benefits. Firstly, it provides an intuitive set of metric physical properties we can set to achieve a desired trajectory, such as the accelerations and the drag coefficients. Secondly, it naturally produces smooth trajectories as accelerations are integrated over time, with the notable exception of the simple collision system described in more detail below. Finally, although not provided in this dataset, it provides physically based motions from which IMU accelerometer measurements could be derived (for example via spline fitting) which could in future prove useful for Visual-Inertial systems.

We initialise the camera and look-at point by sampling from a uniform random distribution within the bounding box of the scene, ensuring they are less than 0.5m apart. As not all scenes are convex, it is possible to initialise the starting points outside of a layout, for example in an 'L'-shaped room. To reduce the frequency of invalid trajectories such as this we employ three heuristics. The first is to restart the simulation if either body leaves the bounding volume. This is because in the case it is initialised outside of the layout there would exist no collision surfaces to prevent it from exiting the bounding volume completely. The second is that within the first 500 poses (a 'burn-in' period) at least 10 different object instances must be visible. This prevents trajectories external to the scene layout with only the outer wall visible. Finally we require that both bodies have ventured more than 1.0 m from the starting position during the burn-in period. This is to prevent situations where the bodies are trapped in certain layouts; for example in the cavity between two wall planes.

We use simple Euler integration to simulate the motion of the bodies and apply acceleration vectors to them independently. Each body is initialised with a position, $\mathbf{p}$, in the manner described above and a velocity, $\mathbf{v}$, initialised to zero. We sample from a uniform spherical distribution by first sampling from a 3-dimensional Gaussian,

$$\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{5.1}$$

and normalising $\mathbf{g}$ to be on the unit sphere. This is then scaled by an acceleration constant, a, which we set to $2.5\,\mathrm{m\,s^{-2}}$, to calculate the random acceleration vector, $\mathbf{a}$:

$$\mathbf{a} = a\frac{\mathbf{g}}{||\mathbf{g}||}. \tag{5.2}$$

We also include drag to dampen fast motions. We roughly model this as air drag on a smooth sphere of radius 0.3 m and mass of 1 kg. With a cross-sectional area $A = 0.09\,\mathrm{m}^2$, drag coefficient $C = 0.1$, and air density $\rho = 1.2\,\mathrm{kg\,m^{-3}}$, the acceleration due to drag becomes:

$$\mathbf{d} = -\frac{\mathbf{v}}{2||\mathbf{v}||}\rho AC||\mathbf{v}||^2. \tag{5.3}$$

We use simple semi-implicit Euler integration over a timestep, τ, which here we set to $\frac{1}{60}$s for the shutter open exposure time and $\frac{7}{300}$s for the shutter closed time to fill the remainder of the 0.04s period needed for 25Hz operation:

$$\mathbf{v}_{t+1} = \mathbf{v}_t + (\mathbf{d} + \mathbf{a})\,\tau. \tag{5.4}$$

We also limit the maximum speed of the body to a threshold, $s_{\max}$, which we set to 3 m/s in our dataset:

$$\mathbf{v}_{t+1} = \begin{cases} s_{\max}\frac{\mathbf{v}_{t+1}}{||\mathbf{v}_{t+1}||}, & \text{if } ||\mathbf{v}_t|| > s_{\max} \\ \mathbf{v}_{t+1}, & \text{otherwise.} \end{cases} \tag{5.5}$$

After this we then update the position with the capped velocity:

$$\mathbf{p}_{t+1} = \mathbf{p}_t + \mathbf{v}_{t+1}\tau. \tag{5.6}$$

Finally, to avoid collisions with the scene or objects we render a depth image using the z-buffer of OpenGL. We render from a virtual camera looking in the direction of the body's velocity vector and if there is a depth reading smaller than the body's collision radius (0.3 m) then we consider a collision to have occurred. On a collision the velocity of the body is simply negated in a 'bounce', which simplifies the collision by ignoring the actual surface normal and assuming that the surface normal is always equal to the negative velocity vector. The end result of this process is the two-body trajectory that has already been visualised in Figure 5.1. Listing 5.1 shows a sample trajectory file defining the scene and camera trajectory that is then passed to the rendering engine.

5.6 Rendering RGB Frames

The rendering engine used was the Opposite Renderer[5] [Pedersen, 2013], a flexible open-source ray-tracer built on top of the NVIDIA OptiX framework. We added certain extra features such as Phong shaded specular materials, ground truth materials, and multiple photon maps which can be stored in CPU memory and swapped into the GPU as they are too large to keep entirely on the device. Although there were other open-source alternatives that we considered such as POVRay, Blender, and OpenGL each one had their own limitations. For instance, although POVRay is able to use multi-threading on the CPU to improve performance, it does not have GPU support and we had a number of GPUs available for rendering purposes. It is not easy to render high quality visual artefacts such as global illumination, caustics, reflections or transparency in OpenGL and in our initial experiments we did not find Blender as flexible as OptiX for customised rendering.

[5] http://apartridge.github.io/OppositeRenderer/

Listing 5.1: Partial scene layout textfile output after trajectory generation.

```
layout_file:./bedroom/bedroom3_layout.obj
object
03938244/218f86362028a45b78f8b40f4a2ae98a
wnid
03938244
scale
0.416493
transformation
0.999238   -0.00604157  -0.0385491    1.46934
0.00627241  0.999963     0.00587011  -0.0346129
0.0385122  -0.00610744   0.999239    -1.00603

object
03938244/ac2477b9b5d3e6e6c7c8ce3bef5c2aa9
wnid
03938244
scale
0.169709
transformation
 0.505633    0.123627   0.853845   3.57641
-0.00155019  0.989809  -0.142395  -0.0223919
-0.862747    0.070676   0.500672  -0.377113
....

# Poses come in alternating pairs. With shutter open
# on the first line then shutter close on the next.
# Each line has a timestamp in seconds as well as
# the camera position and look at position both defined
# in world coordinates. The layout is as follows:
# time cam_x cam_y cam_z lookat_x lookat_y lookat_z

# frame rate (Hz):      25
# shutter duration (s): 0.0166667

0.0000 -2.157 1.234 2.384  -0.5645 2.491 0.5848
0.0167 -2.157 1.233 2.384  -0.5646 2.490 0.5847

0.0400 -2.156 1.232 2.384  -0.5647 2.489 0.5843
0.0567 -2.156 1.232 2.384  -0.5648 2.489 0.5841
```

We do not have strict real-time constraints to produce renders, but the scale and quality of images required does mean the computational cost is an important factor to consider. OptiX is specifically designed for rendering on the GPU and it is able to make good use of the computational capacity offered by modern day GPUs. More recently NVIDIA's new RTX family of GPUs[6] have included within them hardware designed specifically to improve the computational performance of ray tracing frameworks such as OptiX. They aim to allow real-time ray tracing applications and in future these improvements will allow similar approaches to ours to have improved performance. This framework also provides us with significant flexibility with our rendering pipeline, enabling us to obtain ground truth information of various kinds

[6]https://developer.nvidia.com/rtx/raytracing

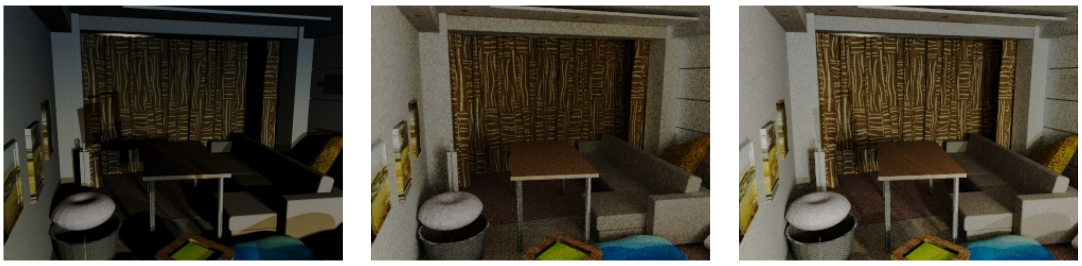

(a) Direct & specular (b) Surface radiance (c) Combined

Figure 5.7: Comparison of direct and indirect photon map gathered light.

such as depth and object instance indices in a convenient manner. Moreover, in future it could also allow for more complicated BRDF surface properties to be easily modelled.

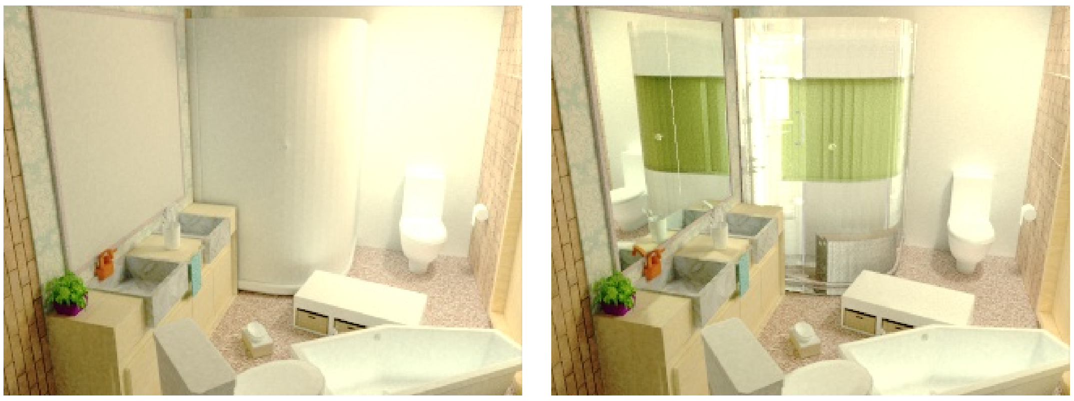
5.6.1 Photon Mapping

We use a process known as photon mapping to approximate the rendering equation. Our static scene assumption makes photon mapping particularly efficient as we can produce photon maps once for a given scene and query them repeatedly throughout the camera trajectory. A good tutorial on photon mapping by [Jensen and Christensen, 2000] is available for more details.

As a quick summary, this technique works via a two-pass process. In the first pass, simulated photons are emitted from light sources accumulating global illumination information and storing this information in a photon map. In the second pass radiance information from this photon map is gathered along with direct illumination from light sources and specular reflections using ray-tracing to produce the final render. These separate and combined images can be seen in Figure 5.7. Normal ray-tracing allows for accurate reflections and transparency renderings, but photon mapping provides an efficient global illumination model that also approximates indirect illumination, colour-bleeding from diffuse surfaces, and caustics. Many of these effects can be seen through the transparent shower enclosure in Figure 5.8.

5.6.2 Rendering Quality

Rendering over 5M images requires a significant amount of computation. We rendered our images on between 4-12 GPUs for approximately one month. An important trade-off in this calculation is between the quality and quantity of images. Fig-



(a) No reflections & transparency (b) With reflections & transparency

Figure 5.8: Ray tracing allows for accurate reflections and transparency effects.

ure 5.9 shows two of the most important variables dictating this balance within our rendering framework, the number of rays sampled per pixel and the number of photon maps generated. The multiple rays per-pixel act to smooth aliasing effects while fewer photon maps lead to a blotchy appearance with gaps between the light patches within the radius of the photons.

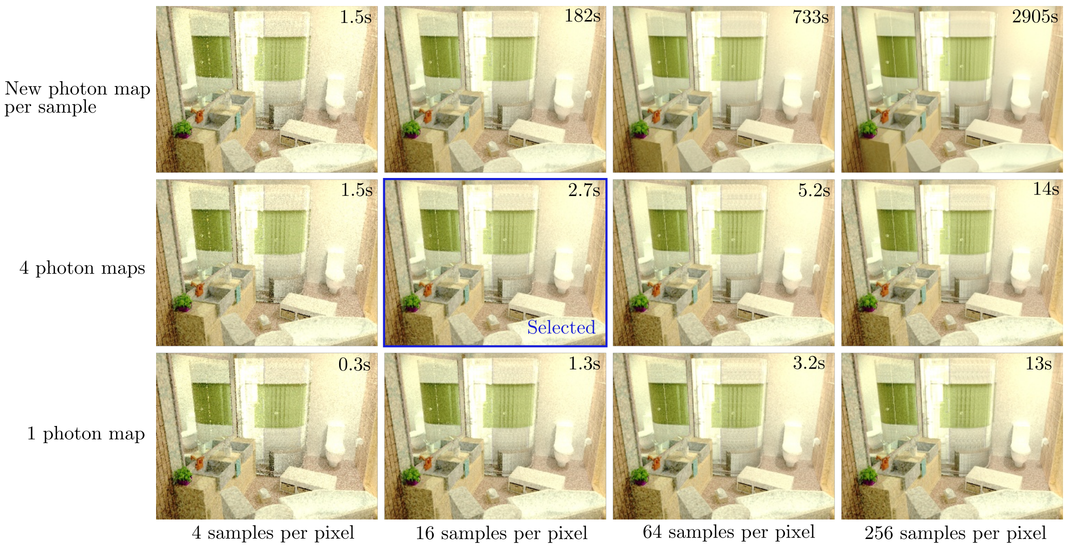


Figure 5.9: Illustration of the trade-off between rendering time and quality. The top-right is at maximum quality but took over 48 minutes to complete.

A single pre-calculated photon map stores approximately 3M photons and requires 4GB of memory. As our GPU memory was not large enough to store sufficient photon maps to attain the desired rendering quality we store additional photon maps in main memory and swap them to the device. The computational cost of this is the amortised photon map computation cost across a trajectory as well as the host-device transfer overhead. If more than 8 photon maps are required this exceeds the available 32GB of main memory on our computer and we must either store to disk or recompute a new set of photon maps for each frame in a trajectory. This is why in the first row and after the first column of Figure 5.9 it takes so much more time as we must include the cost of generating new photon maps.

As a compromise between the computational expense per-image and the desired image quality, we chose to render our dataset with 16 samples per pixel and 4 photon maps per scene. At this setting it takes approximately 3 s on an NVIDIA GTX 1080 GPU to produce a single frame (excluding the one-off photon map calculation cost). Unfortunately at this setting some aliasing and photon map artefacts are still visible, and they can be seen in the final dataset.

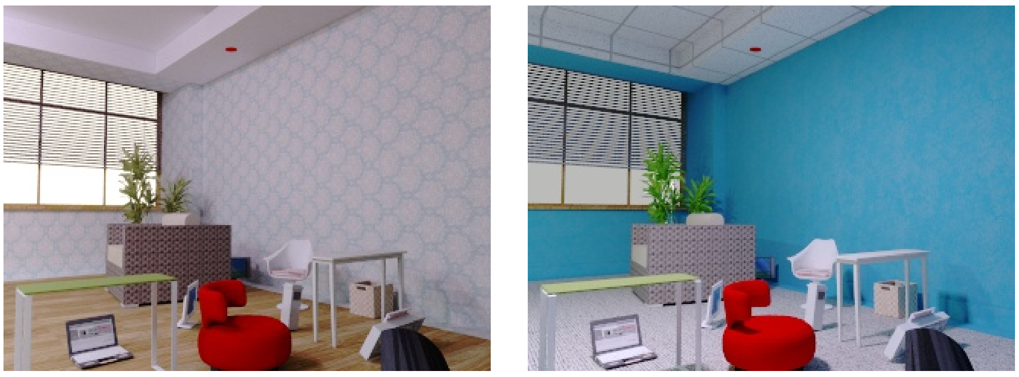
5.6.3 Random Layout Textures and Lighting

To improve the variability within our 57 layouts, we randomly assign textures to each of their constituent components. Each component of a layout has a material type, and for each material type we created a suitable texture library from which a texture can be sampled. For example, we have a large number of different seamless wall, floor, and curtain textures. As well as this, we add random lighting to the scene. Between 1 and 5 lights are randomly added to each scene. We have two types of lights, spherical orbs, which serve as point light sources, and parallelograms which act as area lights. We randomly pick the hue and power of each light and then add them to a random location within the scene. We bias this location to be within the upper half of the scene to mimic light placement in most real-world scenes.

This texture and lighting randomisation approach allows identical geometric layouts to result in renders with quite different visual characteristics as shown in Figure 5.10. In this work we have only rendered a single version of each layout, but the availability of such pairs could prove an interesting facet for training and validating invariance to such differences in competing algorithms.

5.6.4 Camera Model and Motion Blur

Our virtual camera is a simple global shutter pinhole model with a horizontal Field of View (FoV) of 60° and vertical FoV of 45°. This is relatively close to the Kinect's camera intrinsics, but at the lower resolution of 320×240. In order to make sure the



(a) Version 1 (b) Version 2

Figure 5.10: Random lighting and background textures allows the same scene geometry and semantics to produce numerous renderings with quite different appearances.

rendered images are a faithful approximation of real-world images, we also apply a non-linear Camera Response Function (CRF) that maps the irradiance to quantised brightness values. We use a static CRF shown in Figure 5.11, though it would be straightforward to randomise these parameters.

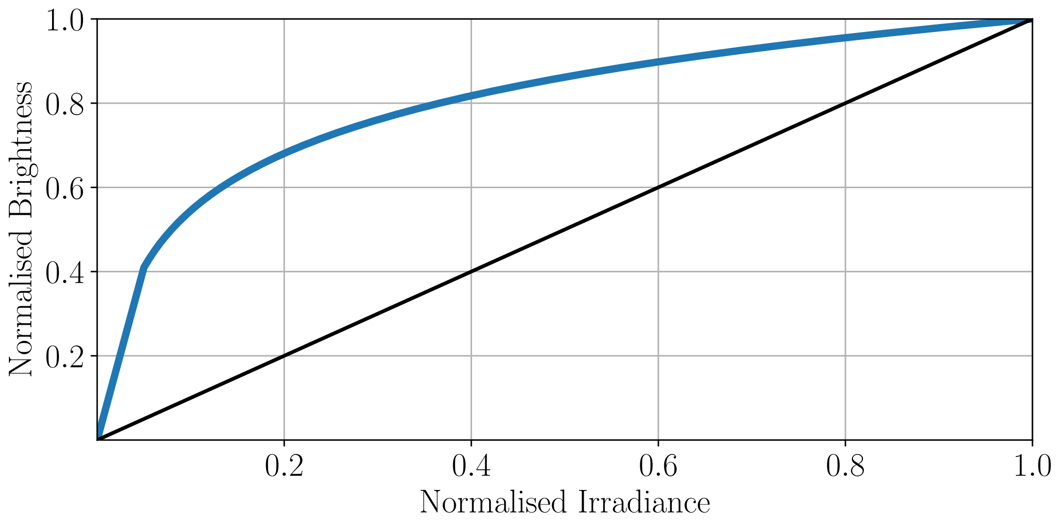
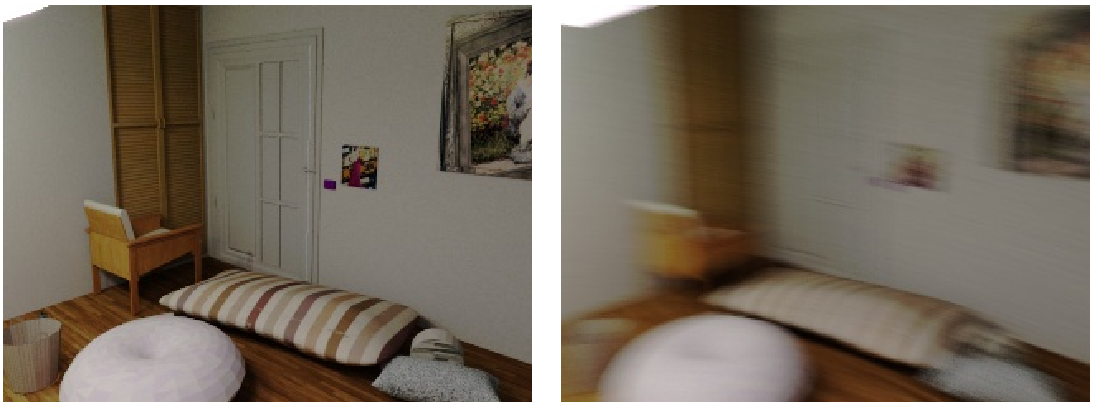


Figure 5.11: The Camera Response Function used by our renderer.

For fast motion we integrate incoming rays throughout a shutter exposure to approximate motion blur. This can be efficiently performed within the rendering process by changing the poses from which each of the supersampled rays are emitted and then properly integrating these irradiance values rather than for example averaging RGB values after rendering.

To calculate the motion blur we draw linearly interpolated lines between the camera position and look-at position at both shutter open and shutter close. Then when rendering we uniformly sample a value in the range (0,1) to define an interpolated camera and look-at pose and then render from those sampled poses. For an example rendering using this technique see Figure 5.12. The motion blur does not affect the ground truth output of depth or instance segmentations. For those renders we set the pose to be the exact midpoint of the shutter exposure.



(a) Without motion blur (b) With motion blur

Figure 5.12: An example of the effect of the motion blur artefact. In this example the camera speed has been increased in order to better illustrate the effect but in the actual dataset itself the effect is generally much less pronounced.

5.7 Experiments

We test the value of SceneNet RGB-D as a training set for semantic segmentation using the real-world NYUv2 and SUN RGB-D datasets as our benchmarks. More specifically, we compare the performance of three different pre-trained network weights on the task of per-pixel semantic labelling. The three weight initialisations are: a network trained from scratch with initialisation proposed by [He et al., 2015], a network (originally initialised with [He et al., 2015]) pre-trained on the 5M synthetic RGB images from the SceneNet RGB-D dataset, and a network initialised with the VGG-16 ImageNet weights. As the VGG-16 ImageNet weights are for a classification task, only the feature-encoder layers (the bottom layers which process the raw input image) can be initialised with them; the layers above this are randomly initialised with the scheme proposed by [He et al., 2015].

The comparison of ImageNet vs. synthetic RGB-only is particularly challenging as the ImageNet weights are trained using 1M real-world images, just as our final test datasets are drawn from the real-world. The question we seek to answer here is whether the task-specific information available in our dataset of indoor scene classes and per-pixel ground truth labelling (instead of simple classification tags of unrelated semantic entities) combined with 5$\times$ more images is enough to counter-balance the advantages of real-world images.

We also experiment with another advantage that our synthetic dataset provides; a wider variety of potential input modalities. We modify the network architecture to include a depth-channel and train this from scratch using the SceneNet RGB-D dataset. We compare this (after fine-tuning) to training from scratch on the real-world dataset alone but we do not directly compare the depth network against ImageNet pre-training for a number of reasons. First, ImageNet does not provide depth data so there is no depth channel for publicly available weights to directly compare against. Second, in the RGB-D network architecture described below we split the feature maps evenly between depth and RGB due to memory constraints; this unfortunately prevents a direct mapping of VGG-16 weights into even the RGB-only part of the network. For both the NYUv2 and SUN RGB-D datasets we choose the same 13 class semantic mapping defined by [Couprie et al., 2013] used in Chapter 4. We manually map each of the WordNet indices in our dataset to one of those 13 semantic classes.

5.7.1 Network Architectures

We choose the straightforward U-Net [Ronneberger et al., 2015] architecture as the basis for our experiments, with the slight modification of applying Batch Normalisation [Ioffe and Szegedy, 2015] between each convolution and non-linearity. It can be seen visualised in Figure 5.13. Our inputs are RGB 320$\times$240 images and our output is a tensor of 320$\times$240$\times$14 class probabilities (the first class being ‘missing ground truth label’ in SUN RGB-D and NYUv2 and ignored in the loss function for training purposes). The RGB-only network contains 22M free parameters.

To accommodate a depth channel in the RGB-D CNN we modify the U-Net architecture to have an additional column of depth-only convolutions. The second half of U-Net remains almost unchanged; we simply concatenate the output of both the depth and RGB feature maps during the upsampling portion of the CNN. We maintain the same number of feature maps in the encoder half of the CNN, but split

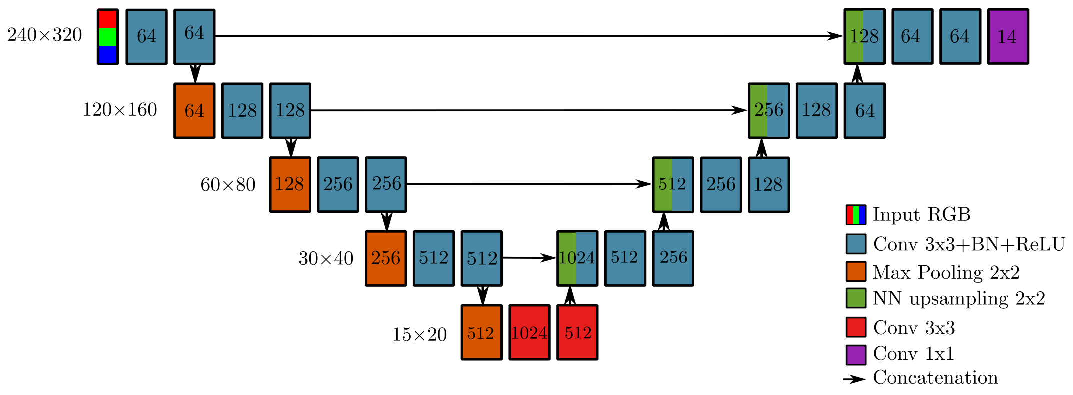


Figure 5.13: The schematic for the RGB-only architecture based on the U-Net [Ronneberger et al., 2015] concatenation approach to skip layers.

the channels evenly between depth and RGB, *i.e.* the first 64-channel RGB convolution becomes a 32-channel RGB convolution and 32-channel depth convolution. Finally, to maintain approximately consistent memory usage and batch sizes during training, we drop the concatenation skip connection at the lowest resolution feature map. A diagram of this architecture is shown in Figure 5.14. Overall these changes reduce the number of free-parameters to 17.2M. This is only one architectural choice for including depth and other schemes such as FuseNet proposed by [Hazirbas et al., 2016] could be explored in future.

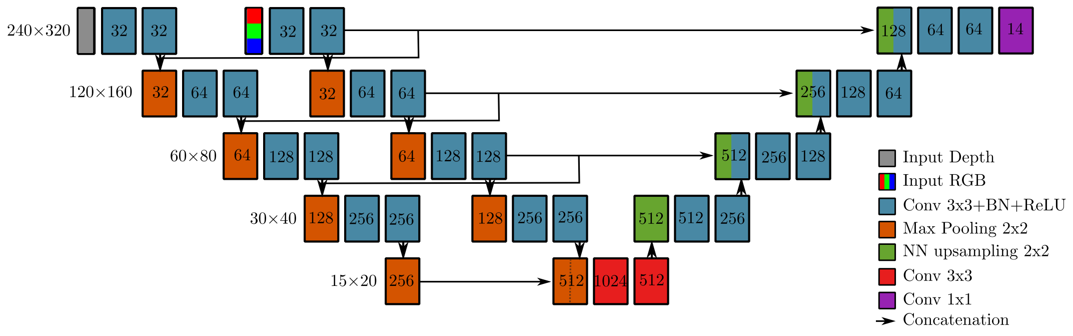


Figure 5.14: The schematic for the modified U-Net architecture that allows a depth channel to be included in an additional column of inputs.

5.7.2 Training

Our network implementation is built within the Torch7 [Collobert et al., 2011] framework and trained on a multi-GPU system. We train with the largest batch-size that would fit in memory. Depending on the network architecture the batch-size varied from 25 to 30 images. For all experiments the learning rate was initialised at 0.1, and scaled by 0.95 after every 30 epochs. Our networks pre-trained on SceneNet RGB-D were trained for less than the 30 epochs at which the learning rate would be scaled so both maintained a constant learning rate throughout training. The RGB CNN was pre-trained for 15 epochs which took approximately 1 month on 4 NVIDIA Titan X GPUs, and the RGB-D CNN was pre-trained for 10 epochs, taking 3 weeks.

We use SGD with momentum of 0.95 for optimisation and no weight regularisation. We use the standard train/test split on the NYUv2 and SUN RGB-D datasets and for all fine-tuning experiments. We perform early stopping and give the validation performance on the test set in the results. Generally fine-tuning required ≈50 epochs.

5.7.3 Results

The results of our experiments are summarised in Tables 5.2 & 5.3. For RGB we see that in both datasets the network pre-trained on SceneNet RGB-D outperformed the network initialised with VGG-16 ImageNet weights on all three metrics. For the NYUv2 the improvement was +8.4%,+5.9%, and +8.1% for the class average, pixel average, and mean IoU respectively, and for the SUN RGB-D dataset the improvement was +1.0%,+2.1%, and +3.5% respectively. This result suggests that a large-scale high-quality synthetic RGB dataset with task-specific labels can be more useful for CNN pre-training than even a quite large (≈1M image) real-world generic dataset such as ImageNet, which lacks the fine-grained ground truth data (per-pixel labelling), or domain-specific content (indoor semantics). The trend between the two datasets is also clear, the improvement from pre-training on the synthetic data is less significant when fine-tuning on the larger 5K image SUN RGB-D dataset vs. the 795 image NYUv2 dataset.

The inclusion of depth as an additional input channel resulted in a significant performance improvement over RGB-only in both datasets (+6.2%,+5.6%,+6.4% in the NYUv2, and +5.5%,+5.0%,+6.0% in the SUN RGB-D for the class average, pixel average, and mean IoU respectively). When compared against training from scratch for the depth channel, pre-training showed a clear performance improvement. We found training the architecture from scratch on the depth modality challenging, taking longer to converge (300 epochs) and with infrequent classes, such as books and TV showing particularly poor results in both datasets.

NYUv2: 13 Class Semantic Segmentation

Pre-training	bed	books	ceiling	chair	floor	furniture	objects	painting	sofa	table	tv	wall	window	class avg.	pixel avg.	mean IoU
No pre-training RGB	42.6	11.8	57.5	15.5	76.1	55.5	37.9	48.2	17.9	18.8	35.5	78.4	51.1	42.1	55.8	29.0
ImageNet	45.7	29.4	56.4	35.9	84.1	51.3	43.4	58.6	24.4	26.2	18.8	80.8	63.8	47.6	60.2	33.7
SceneNet RGB	54.7	**30.0**	65.0	48.3	86.2	59.7	52.5	**62.1**	50.2	32.5	40.5	82.4	64.5	56.0	66.1	41.8
No pre-training RGB-D	58.2	1.7	**75.5**	48.7	**94.8**	54.6	47.6	38.2	42.2	32.8	23.2	82.5	43.0	49.5	63.4	36.9
SceneNet RGB-D	**69.4**	22.7	71.1	**63.8**	**94.8**	**64.4**	**56.8**	61.2	**68.9**	**41.0**	**43.1**	**84.3**	**66.9**	**62.2**	**71.7**	**48.2**

Table 5.2: **NYUv2 validation set results**: Segmentation performance using U-Net architectures described above. The CNN that was pre-trained on SceneNet RGB-D images outperformed the same CNN trained from scratch and with VGG ImageNet weights after fine-tuning. The inclusion of depth data also improved the classification performance over RGB-only, and SceneNet pre-training again improved performance over the baseline approach. After the listed form of pre-training, all networks are then fine-tuned on the NYUv2 train set. All evaluations were performed at 320×240 resolution.

SUN RGB-D: 13 Class Semantic Segmentation

Pre-training	bed	books	ceiling	chair	floor	furniture	objects	painting	sofa	table	tv	wall	window	class avg.	pixel avg.	mean IoU
No pre-training RGB	47.0	26.9	50.0	57.7	88.9	38.4	31.2	39.7	43.0	55.8	18.2	84.6	61.2	49.4	68.9	36.5
ImageNet RGB	56.8	**42.2**	58.0	64.7	88.6	42.9	**49.2**	**59.1**	59.2	51.9	22.6	84.1	59.2	56.8	71.2	40.3
SceneNet RGB	56.4	29.0	66.7	68.9	90.1	53.0	43.4	50.4	51.6	60.1	28.7	83.5	**69.0**	57.8	73.3	43.8
No pre-training RGB-D	**69.0**	19.3	68.0	68.8	**94.3**	46.7	37.8	41.6	55.8	59.0	5.7	86.7	50.5	54.1	73.9	41.4
SceneNet RGB-D	67.2	33.8	**76.2**	**71.6**	93.9	**55.4**	46.5	56.1	**63.1**	**72.8**	**35.8**	**88.2**	61.7	**63.3**	**78.3**	**49.8**

Table 5.3: **SUN RGB-D validation set results**: Similar to the NYUv2 results, the CNN that was pre-trained on SceneNet RGB-D images again outperformed the other two weight initialisations after fine-tuning. However the improvement over the ImageNet weights is less significant here on the approximately $5\times$ larger SUN RGB-D dataset. The inclusion of depth data again improved the classification performance over RGB-only. After the listed form of pre-training, all networks are then fine-tuned on the SUN RGB-D train set. All evaluations were performed at 320×240 resolution.

5.8 Limitations

There is still significant future work that remains to be done in this area. For the pipeline itself, one obvious limitation is that it is not suitable for dynamic scenes as the system has been developed around a static scene to take advantage of efficient rendering techniques. Another limitation is due to the lack of a curated dataset of high quality metrically-scaled object models with detailed physical meta-data. As mentioned in Figure 5.4, our automated systems mistakenly found mailboxes from ShapeNets when searching for the object category 'box.' This unfortunately led to large numbers of mailboxes in indoor scenes. At present even a synthetic dataset requires significant manual intervention in cases such as this to prevent mistakes.

The lack of metric scales in ShapeNet meant that we had to resort to automated methods of scaling objects. Many objects in modern domestic environments are designed and built using highly detailed and accurate computer models. Given this data exists, in future it may become possible to use it for research purposes. Accurate physical information of objects not only can improve the visual appearance of objects in the scene, but also allows interactive scene dynamics to be properly modelled. Some work is being done in this direction with the photorealistic InteriorNet [Li et al., 2018] dataset which uses industrial object model repositories, with accurate metric scales and physical meta-data to simulate traces of daily life.

Here we have aimed to produce a dataset that is as close as possible to the domain found in the real-world. While tuning the quality of the dataset, we have largely relied on a human assessment of what appears 'more' realistic. However, in our experiments the synthetically trained CNN still had to be fine-tuned on real images to perform well in the real-world. This suggests that, despite our best efforts, there remain significant differences between our synthetic approximation and the real-world. Further research is needed to analyse and verify what the most important discrepancies are for the purposes of CNN training. It could be that the rendering engine itself failed to produce images of sufficient quality, either by missing important image artefacts entirely (*e.g.* an explicit noise model) or as a result of the compromises taken to reduce the computational expense of rendering. It could also be that the quality of the underlying assets used was insufficient to properly mimic real-scenes.

Generative Adversarial Networks (GANs) [Goodfellow et al., 2014] may be particularly useful for this problem. GANs make use of both a generative and a discriminative model which are trained together in an adversarial manner. The generative model is trained to produce samples that mimic the distribution of an input dataset, while the discriminative model learns to detect whether a given sample is real or generated. This technique could be applied to synthetic datasets in a number of ways. One approach could be to improve the existing dataset by conditionally translating images to better mimic a real-world dataset. The work of [Zhu et al., 2017] on Cycle GANs allows domain translation without paired images in both domains. This would allow the use of any large real-world dataset to serve as the target domain for translation.

Another route is to circumvent the inherent difficulties involved with producing such a faithful approximation entirely. [Ren and Lee, 2018] used images from SceneNet RGB-D and SUN-CG to train an adversarial network to detect whether generated CNN *features* were produced from a real or a synthetic image. In this way the CNN itself learns to minimise the domain differences between the two datasets and so improve domain transfer. A similar non-adversarial approach is to jointly optimise for a given task as well as the domain transformation itself [Sener et al., 2016]. A very different approach to circumvent the requirement of a faithful real-world simulation is called domain randomisation [Tobin et al., 2017]. Instead of aiming for a very realistic synthetic training set, domain randomisation seeks to make the training domain so varied that the model can directly cope with the real-world as just another possible variation. Regardless of the manner in which the problem is approached, it is clear that mitigating domain transfer and the 'reality-gap' is a key area of research that is yet unsolved and will need to be better understood to allow fully synthetic approaches to become useful in real-world applications.

5.9 Conclusion

In this chapter we have approached the problem of segmentation dataset generation for indoor scenes by using computer rendered synthetic data. This has enabled the production of a large-scale dataset of reasonable quality and at low cost. We have outlined a pipeline which provides a solution to problems such as the generation of physically plausible scene layouts and automated randomised camera trajectories.

The randomness inherent in our pipeline also allows for a continuous stream of unseen training examples to be generated. In the future, we believe it is likely that the generation of synthetic training data and the training procedure for the models themselves will become more tightly interleaved. An example of this is curriculum learning [Bengio et al., 2009] where simpler subtasks are used as pre-training in order to prime the model with features useful for more complicated tasks.

Our approach to trajectory generation was simple and provided an unlimited stream of different camera trajectories. It could in future be tailored for the type of trajectory that is expected in the final dataset. However in the present form it lacks certain realistic characteristics of a hand-held camera trajectory. The work of [Li et al., 2018] expanded upon the trajectory generator used here to include different styles of camera movement and also explored adding a learned camera shake model to improve the realism of the otherwise artificially smooth trajectories. A complimentary approach that could be explored is to match the semantic distribution of selected views with those seen in a source dataset [Genova et al., 2017] in order to produce tailored training data. Although that approach was for still-frames only, the same idea could be applied to camera trajectories.

The results of our experiments show an RGB-only CNN pre-trained from scratch on synthetic RGB images can improve upon the performance of an identical CNN partially initialised with ImageNet weights. This experiment was only performed for a single style of network architecture and it may be the case that alternative models or training methods could make better use of the pre-trained ImageNet weights. However the experiment does illustrate the value synthetic datasets can bring to real-world problems at relatively low cost compared to capturing and annotating a real-world dataset. The additional performance improvement achieved by including the depth also serves to highlight the importance of the flexibility and ease with which synthetic data can produce alternative modalities.

Certain capabilities of the dataset as it stands remain to be explored. As discussed in Chapter 4 (Section 4.5) the importance of different viewing poses for certain object classes and clutter needs further analysis and requires the variety of views and ground-truth information available in synthetic datasets such as SceneNet RGB-D. The availability of smooth temporal trajectories with camera pose data provides multi-view correspondences, and lends itself to the exploration of SLAM systems and CNN architectures with a more explicit temporal component, such as the work by [Xiang and Fox, 2017] on Data Associated Recurrent Neural Networks.

As the dataset has been made publicly available, its benefits are not limited to the experiments that have been performed here. It has since enabled other researchers to explore a variety of problems. The scale and availability of ground truth depth has allowed it to be used as training data for single view depth prediction [Chen and Deng, 2018]. The inclusion of image artefacts such as motion blur has led to its use for pre-training networks in order to mitigate problems specific to robot vision [Balloch and Chernova, 2017]. The dataset's focus on video trajectories rather than still images led to its use in deep learned ego-motion induced view prediction [Shamwell et al., 2017] in novel deep learned SLAM systems such as CodeSLAM [Bloesch et al., 2018].

Chapter 6

Fusion++

Contents

6.1 Introduction

The work in Chapter 4 highlighted important limitations of what could be achieved with SemanticFusion's representation for dense 3D semantic mapping. Interpreting

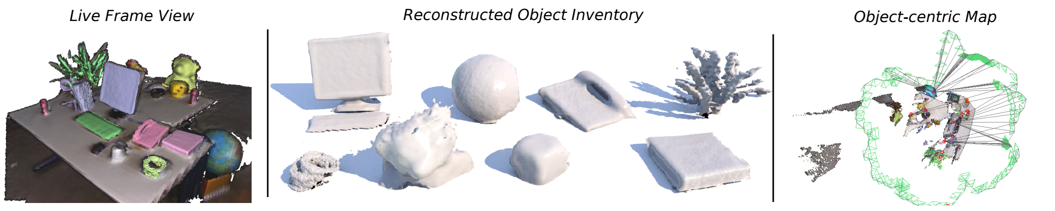


Figure 6.1: Fusion++ result on the `fr2_desk` sequence [Sturm et al., 2012]. **left**: live view combining instance detections (in pastel colours) overlayed on the throwaway background TSDF. **middle**: a rendering of the discovered objects in the map. **right**: the object-level pose-graph and camera trajectory.

and using a dense map with millions of independent semantic probability distributions requires a further level of processing before the map can be used in an intuitive manner. It does not allow task-relevant queries such as 'How many chairs do we have in the conference room?' to be directly answered. This problem could be approached by clustering semantically similar surfels into 'instances' but such an approach would fail when multiple semantically similar instances are in close proximity, and also requires *semantic* understanding as a pre-requisite for instance segmentation. Another problem with SemanticFusion is that it lacks any form of instance grouping, so it is possible to produce highly inconsistent semantic maps with different surfaces of the same object being labelled with different semantic labels. As the surfels in the map are completely independent there is no means to pool the semantic information of the object from opposing sides.

In this chapter we make the paradigm shift away from dense semantic annotation to object-level (or object-oriented) instance mapping. Unlike many dense reconstruction systems [Newcombe et al., 2011b, Whelan et al., 2012, Zhou et al., 2013, Whelan et al., 2015b, Choi et al., 2015, Dai et al., 2017b] we make no attempt to keep a dense representation of the entire scene. Our persistent map consists of only reconstructed object instances that are discovered and segmented using object detection and instance masks to form a scene inventory (illustrated in Figure 6.1). Each object is contained within a separate TSDF volume [Curless and Levoy, 1996], allowing each one to have a different, suitable, resolution. We believe this is a natural and efficient way to represent much of the data that is important for robotic scene understanding and interaction and it is also highly suitable as the basis for human-robot communication.

Combining instance segmentation with online SLAM and TSDF reconstruction created new problems which required a number of contributions. The first issue is that if one were to simply reconstruct an object by only fusing depth measurements from the segmentation mask into the volume, useful measurements about surface geometry near the object would be discarded. Due to the variability in instance mask predictions, the predicted segmentation mask could increase in size and suddenly new areas would have to be reconstructed from scratch. It also causes problems when the segmentation mask decreases in size, as previously reconstructed surface geometry persists and would require some other mechanism for removal.

To solve this problem we propose reconstructing all of the geometry within the TSDF volume, regardless of the instance mask. To segment the object we instead use an incrementally fused 3D foreground probability mask encoded in each voxel to denote surfaces that belong to the object's foreground. We found that fusing the 2D instance masks using a multiplicative Bayesian update scheme, such as that of SemanticFusion, was unstable, as a single poor mask prediction with a false negative of very low probability would occasionally destroy the entire foreground mask. To alleviate this we present a more robust approach which views the binarised instance mask as the result of a binomial trial in a manner analogous to coin flips and use a Beta distribution conjugate prior to calculate the expectation a given voxel corresponds to the object's foreground.

A second issue encountered was related to the object-level pose-graph. The use of an object-level pose-graph allows a globally consistent map of instances to be produced on repeated loops of a scene, but to produce the edge measurements for each object in the pose-graph it is necessary to track individual objects using ICP. In SLAM++ this was possible because they were operating on a small database of pre-reconstructed objects with geometry well suited for tracking. In Fusion++ the greater variety of object geometries discovered as well as noise from the input depth map during the initial stages of online reconstruction, led ICP tracking of individual objects to fail to converge. In order to calculate edge constraints for each object, we therefore develop a more robust approach which first performs ICP on the complete scene geometry and then subsequently partitions the residual error for each object instance. For each partition we perform a single Gauss-Newton step to ensure that the final measurement coincides with the minimum of the object's quadratically approximated error function.

Splitting the map into separate reconstructed instances not only produces se-

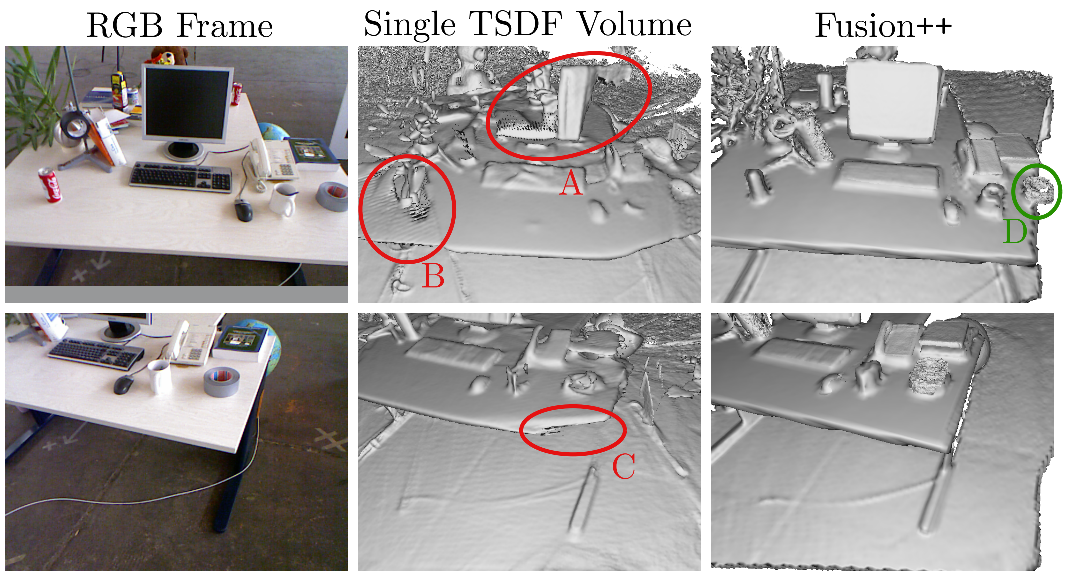


Figure 6.2: A comparison of the reconstruction quality of a single large KinectFusion volume and the Fusion++ system on the `fr2_desk` sequence [Sturm et al., 2012]. When the camera returns to a previously mapped area, the accumulated pose drift quickly degrades the reconstruction quality of the scene with the standard KinectFusion approach. In contrast Fusion++ can perform loop-closure and optimise the camera and instance volume poses to best fit the measurements, rather than eroding surface geometry. **A**: The previous reconstruction of the computer monitor is being eroded on the right, and a new reconstruction is being created on the left to match the current measurements. In Fusion++ the original reconstruction can be seen in the correct pose. **B & C**: New surfaces are being generated for the desk to accommodate the offset measurements caused by drift. **D**: The duct tape instance created on the first loop can be seen in the correct pose for the next loop.

mantically meaningful map entities, but also can improve the reconstruction quality after repeated loops of a scene. If a single large KinectFusion volume is used to reconstruct a large-scene, the accumulated drift on repeated loops leads to the erosion of and duplication of previously mapped surfaces (illustrated in Figure 6.2). This reduces the reconstruction quality and can also lead to severe deterioration in tracking performance. The approach taken here, of combining the high-quality reconstructions possible with TSDFs with the flexibility of a pose graph fixes this by allowing the relative poses of locally reconstructed segments of geometry to be optimised in an online manner and without the complication of performing intra-TSDF deformations or object re-integration.

We aim to produce reconstructions of object instances without strong *a priori* knowledge of the objects present in the scene. There are many ways in which an ‘object’ can be defined and discovered. Unsupervised approaches can look for repeated elements within a scene and temporal cues can be used to segment components of a scene which move together via change detection. In this work we use a pre-trained supervised model called Mask R-CNN [He et al., 2017, Wu et al., 2016b] to provide 2D object detections and instance mask predictions. This means that although the geometric properties of the objects can vary dramatically, no completely new semantic classes of objects are discovered. The definition of an object is thereby modelled by the network weights of Mask R-CNN trained on a selected dataset.

This approach is quite flexible as any task-specific grouping of objects can be used by providing a dataset with the desired grouping and fine-tuning Mask R-CNN with it. The breadth of newly discovered object types that can be semantically grouped is also quite wide. We have found that generic classes such as the NYUv2 dataset’s `other_prop` class allow a wide variety of objects to be reconstructed and segmented without strong semantic cues. In this way, the Mask R-CNN system can rely on ‘objectness’ cues to become a more general object detector which can be used in the current system. Miscellaneous discovered objects could then be filtered or manually annotated and potentially used for a bootstrap training scheme. For object detection we also conservatively filtered for favourable views of an object (*i.e.* fully visible and centred in the image) to improve the instance segmentation quality and TSDF initialisation. Such favourable views are more likely to occur during longer mapping operations by an active agent.

For each object a single semantic probability distribution is maintained and updated. The approach of first grouping and then labelling entities neatly avoids the memory limitation problems inherent in storing a semantic distribution for each voxel independently which necessitates approximation approaches [Cavallari and Di Stefano, 2016c]. Explicitly storing a semantic distribution with thousands of different classes costs barely any additional memory, just a single scalar for each class. The objects also carry with them an ‘existence’ probability which allows a spurious object to eventually be deleted if it is no longer being detected as an object when the model of the map predicts it should be clearly visible.

As part of the theme of this area of research is providing an easy to use developer interface we developed the system itself using a Python front-end. Python has become one of the most popular languages for the machine learning research community in recent years, and new tools such as `pybind11`[1] allow it to operate with C++ and CUDA with very minimal overhead. Computationally intensive tasks in our system such as ray-marching, ICP, and pose graph optimisation are still performed in C++ and CUDA but with a wrapper exposing their core functionality to Python. As our Mask R-CNN is implemented in Tensorflow it naturally forms just another component used within the Python front-end.

The system is designed to operate online but unfortunately it is not yet a truly real-time system. Excluding relocalisation and pose-graph optimisation, the system operates at 10-13Hz on our indoor office scene with the majority of this cost coming from the layered tracking step and raycasting and integration of the numerous object instances. Each visible instance costs another $\approx$1ms for all of these steps, and filtering for the visible set of TSDFs takes approximately 0.3ms per map object (regardless of visibility). Mask R-CNN takes approximately $\approx$250ms for a forward-pass so the system has been designed to perform predictions in a parallel thread so as not to interfere with local tracking. The relocalisation and pose-graph operations are particularly slow at present and use a simple exhaustive search of previous snapshots which is filtered based on object snapshots. We believe that with sufficient software optimisation the methods outlined in this chapter have the capacity to achieve real-time performance on current consumer hardware.

6.2 Related work

For reconstruction, we follow the TSDF formulation of [Curless and Levoy, 1996] and the KinectFusion approach of [Newcombe et al., 2011a] for local tracking. Our approach to object-centric reconstruction is related to the work of [Zhou and Koltun, 2013], where 'points of interest' are detected and the scene reconstructed so as to preserve detail in these areas while distributing drift and registration errors throughout the rest of the environment. In our work we analogously aim to optimise the quality of object reconstructions and allow residual error to be absorbed in the edges of the pose graph. We also aim for an online system so we do not perform the off-line two-pass registration procedure used there, instead selectively integrating depth measurements when in close agreement to the model.

SLAM++ [Salas-Moreno et al., 2013] was an early RGB-D object-oriented map-

[1]https://github.com/pybind/pybind11

ping system which serves as the primary inspiration for the present work. They used point pair features for object detection and a pose graph for global optimisation. The drawback was the requirement that the full set of object instances, with their very detailed geometric shapes, had to be known beforehand and pre-processed in an offline stage before running. [Stückler and Behnke, 2012] also previously tracked object models learned beforehand by registering them to a multi-resolution surfel map. [Tateno et al., 2016] used a pre-trained database of objects to generate descriptors, but they used a KinectFusion [Newcombe et al., 2011a] TSDF to incrementally segment regions of a reconstructed TSDF volume and match 3D descriptors directly against those of other objects in the database. In this work we aim to use Deep Learning to remove the need to build an explicit object database beforehand; instead objects are discovered and reconstructed within the loop of the SLAM system.

A number of approaches to object discovery exist [Collet et al., 2013, Stückler and Behnke, 2013, Choudhary et al., 2014]. Most related to ours is the work of [Choudhary et al., 2014] where they localised the camera in an online manner using discovered objects as landmarks in a pose-graph formulation similar to ours, although they used the point cloud centroid only, whereas in our pose-graph object landmarks provide 6 DoF constraints based on the result of ICP on dense volumes. To discover these objects they clustered points after removing planar regions. They showed that the approach improves SLAM results by detecting loop closures. However, unlike our work they use point clouds rather than TSDFs and do not train an object detector but instead use the unsupervised segmentation approach of [Trevor et al., 2013].

Another approach to object discovery is through dense change detection between successive mappings of the same scene. [Finman et al., 2013] used temporally separated scans of the same scene using Kintinuous [Whelan et al., 2012]. [Ma and Sibley, 2014] performed unsupervised object discovery using a dense TSDF system, introducing objects after an initial reconstruction is created to reconstruct and track objects which are inconsistent with the initial model. More recently [Fehr et al., 2017] used a TSDF approach to perform change detection but on more general dynamic scenes without requiring an initial empty view. Unlike these systems, our system is designed for online use and does not require changes to occur in a scene before objects are detected. These approaches are complementary to our proposed approach for object discovery. They could be used to provide supervisory signals for CNN adaption in the field and enable additional object database filtering mechanisms.

In RGB-only SLAM for object detection, [Pillai and Leonard, 2015] use ORB-SLAM [Mur-Artal and Tardós, 2014] to assist object recognition. They use a semi-dense map to produce object proposals and aggregate detection evidence across multiple views for object detection and classification. MO-SLAM by [Dharmasiri et al., 2016] focused on object discovery through duplicates. They use ORB [Rublee et al., 2011] descriptors to search for sets of landmarks which can be grouped by a single rigid-body transformation. This approach is similar to our relocalisation method, which uses BRISK features [Leutenegger et al., 2011] but augmented with depth.

Very closely related to our approach is the Meaningful Maps work by [Sünderhauf et al., 2017], who proposed an object-oriented mapping system composed of instances using bounding box detections from a CNN and an unsupervised geometric segmentation algorithm using RGB-D data. Although the premise is closely related, there are a number of differences when compared to our system. They use a separate SLAM system, ORB-SLAM2 [Mur-Artal and Tardós, 2015], whereas in our system the discovered object instances are tightly integrated into the SLAM system itself. We also fuse instances into separate TSDF volumes with a foreground/background mask from 2D instance mask detections rather than using point cloud segments.

A number of more recent related pieces of work have also been announced. [Pham et al., 2018] fuse a TSDF of the entire scene and semantically label voxels using a CNN followed by a progressive CRF. To segment instances, instead of fusing native instance detections, they opt to cluster semantically labelled voxels in 3D. This approach, although a natural next-step from dense 3D semantic mapping, is not suitable for object-centric pose graph optimisation and reconstruction as the instances are embedded within a shared TSDF. As already discussed, it also requires *semantic* recognition as a pre-requisite for object discovery.

[Rünz and Agapito, 2018], as in our method, use Mask R-CNN [He et al., 2017] predictions to detect object instances. They aim to densely reconstruct and track moving instances using an ElasticFusion [Whelan et al., 2015b] surfel model for each object, as well as for the background static map. Although using the same prediction model, the approach and goals of these two systems differ substantially. Unlike the present work, they do not aim to reconstruct high-quality objects as pose-graph landmarks in room-scale SLAM. We on the other hand do not currently tackle dynamic scenes and assume all objects to be static during an observation. Clearly there is the long-term potential to combine these two approaches. [Nicholson et al., 2018] also used CNN-based object detection to produce QuadricSLAM, a SLAM system with an object-oriented pose graph. In that work, they used 2D bounding boxes from YOLOv3 to produce object landmarks encoded in the form of quadrics (3D ellipsoids) which provides information about the size, position, and orientation of the objects.

Concurrent work by [Sommer and Cremers, 2018] produced an object-oriented SLAM system combining geometric primitives such as planes and cylinders with known non-primitive objects, detected using Point-Pair features, into a scene SDF. They use this for Bundle Adjustment optimisation as well as scene completion and de-noising. Here we do not consider scene primitives as part of our map, but in indoor environments the ubiquity of planes and other simple primitives lends itself naturally to simpler parametrisations of certain elements of a scene with a fraction of the memory cost. [Salas-Moreno et al., 2013] for example included a simple floor plane in their SLAM++ system. The inclusion of such primitives in the present work is an important area still to be explored.

6.3 Object Detection

Unlike semantic segmentation, object detection does not aim to produce a dense prediction. Instead the location and area of each independent object instance is predicted usually in the form of a classified bounding box. Object detection is frequently split into two distinct phases: object proposal and region classification. In the object proposal step bounding boxes which are likely to contain objects are predicted. This step can even be as simple as an exhaustive sliding window approach using every possible bounding box size and aspect ratio. The region classification then takes each of these windows and either discards them as spurious detections, or classifies them appropriately.

A common final stage in almost all of the methods presented here is to remove heavily overlapping predictions using an approach called Non-Maximum Suppression (NMS). NMS is a greedy algorithm which works from the most to least confident predictions. Predictions with an IoU below a manually selected threshold against all other already accepted predictions are themselves accepted. Predictions which do show a significant overlap with an already accepted (and hence more confident) prediction are filtered out.

An influential approach to using CNNs for object detection is called Regions with CNN Features (or R-CNN) [Girshick et al., 2014]. In that system regions are first proposed using Selective Search [van de Sande et al., 2011] and then the regions are squashed into a 227×227 input image which is fed into AlexNet [Krizhevsky et al., 2012] for feature extraction. The original ImageNet weights of AlexNet were fine-tuned on an $N+1$ classification task, with N object classes plus a background class to filter spurious proposals. Regions with an IoU overlap less that 0.3 with the ground truth were considered false positives for training purposes. Instead of using the output of the Softmax layer directly they then trained $N+1$ separate SVM classifiers on the extracted features. They found that this had slightly better performance than using the softmax layer directly but conjectured that improvements to fine-tuning the CNN would reduce this performance gap.

Some of the earliest research performed in the present work was unpublished research that focused on real-time approaches to object detection. This work is briefly reviewed here as the same techniques were built upon by a number of subsequent object detection systems culminating in the Mask R-CNN [He et al., 2017] system used in Fusion++.

6.3.1 Real-time Object Proposals

R-CNN required approximately 13s to process an image. The Selective Search system takes approximately 2s to form object proposals on the CPU and then 2,000 proposals are processed by the full AlexNet region classifier taking approximately 11s. Some state-of-the-art region proposal techniques available at the beginning of research for this thesis in 2014 such as Multiscale Combinatorial Grouping [Arbeláez et al., 2014] required up to ≈30s to propose regions for a single frame. To explore real-time object detection we looked at two faster object proposal systems. The first was a fast binary SVM proposer called BING [Cheng et al., 2014] and the second was a Deep Learned solution called MultiBox [Erhan et al., 2014].

BING [Cheng et al., 2014] is a very fast object proposal generator based on Binarized Normed Gradients. It operates by resizing an image region to an 8×8 window, and uses the norm of the gradient as a 64 dimensional feature to learn a generic 'objectness' measure in a cascaded Support Vector Machine (SVM) framework. The binarized version requires only a few atomic operations, and has been optimised to work at 300Hz on a consumer CPU. Although fast the performance of BING as part of a full object detection pipeline in areas outside the standard PASCAL VOC 2007 dataset is less impressive. [Hosang et al., 2014] produced a study comparing the effectiveness of different region proposers as part of a detection pipeline. They noted that BING in particular showed a dramatic decline in recall ability at IoU thresholds greater than 50%. As well as this, it also produced the worst mean Average Precision (mAP) results of the 11 proposal methods considered.

An alternative method we explored for improving the efficiency was to use a CNN for both aspects of the object detection system. A forward-pass of AlexNet for a 256×256 image on a GTX Titan Black GPU is ≈24ms. Additionally, the possibility of reusing the same feature maps for both region proposals and region classification would allow even greater performance still. The MultiBox system developed by [Erhan et al., 2014] produces a number of output bounding boxes defined by regressing four offsets l_i from an anchor position (the set of which are calculated via K-means clustering on ground truth object boxes in the training dataset) as well as a confidence value, c_i, for that object detection.

The loss function for training MultiBox requires an indicator variable, x_{ij}, with a 1 denoting the i^{th} anchor's nodes were assigned to the j^{th} ground truth bounding box, g_j, using a bipartite matching algorithm based on Euclidean distance, or 0 otherwise. According to [Erhan et al., 2014], this consistent matching is an important aspect of structuring the problem to ensure convergence. The loss function itself is a simple combination of the Euclidean distance of l_i and the negative log-likelihood of c_i along with a balancing hyper-parameter between the two denoted α:

$$L_{\text{location}}(x, l, g) = \frac{1}{2} \sum_{i,j} x_{ij} ||l_i - g_j||_2^2 \tag{6.1}$$

$$L_{\text{confidence}}(x, c) = -\sum_{i,j} x_{ij} \log(c_i) - \sum_{i} (1 - \sum_{j} x_{ij}) \log(1 - c_i) \tag{6.2}$$

$$L(x, l, g, c) = \alpha L_{\text{location}}(x, l, g) + L_{\text{confidence}}(x, c). \tag{6.3}$$

The base architecture we chose was the GoogLeNet winning entry to the ILSVRC-2014 competition [Szegedy et al., 2015], with the pretrained bottom layers from the classification task[2] fine-tuned and the final fully connected layers trained from scratch (from Gaussian initialization) on the ILSVRC-2014 detection dataset. Training was conducted using Adagrad with a learning rate of 0.005 and an α value of 3. In our experiments the detection rate of the resulting network improved upon both BING and MCG (see Figure 6.3), while only requiring 20ms for inference on a GTX Titan Black GPU.

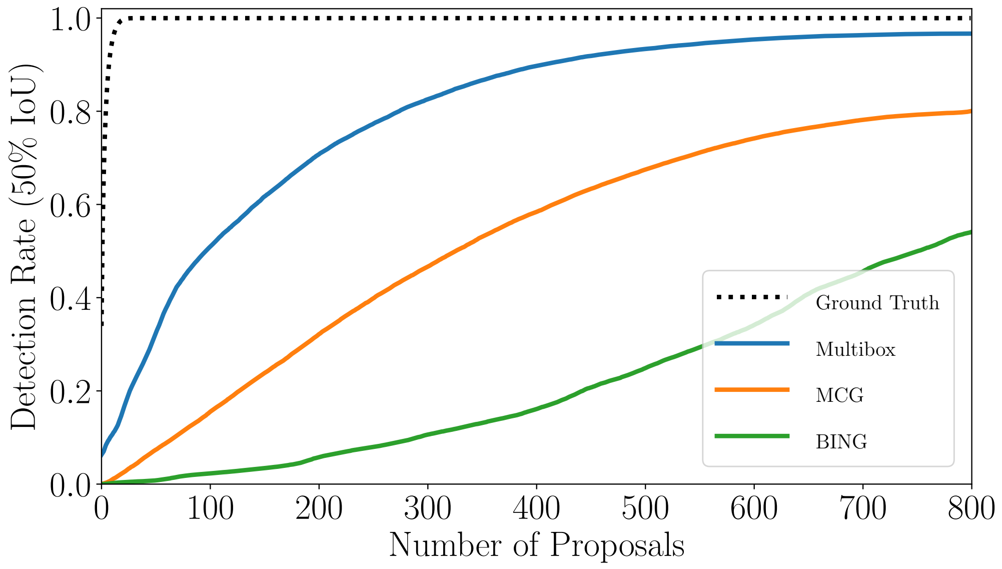


Figure 6.3: A comparison of the detection rates (50% IoU) achieved on the ILSVRC-2014 detection validation set with the three tested methodologies: BING, MCG, and Multibox.

6.3.2 Real-time Region Classification

The second component to the object detection system is region classification. The original R-CNN system proposed [Girshick et al., 2014] simply passed each raw image region patch through the entire CNN to arrive at an output. Although hard to beat in terms of accuracy, this is a hugely expensive procedure. To improve the efficiency of this system, two key points were noted. Firstly, approximately 90-95% of the computation of a CNN is in the lower convolutional layers [Krizhevsky, 2014], and there is a great deal of overlap between the image patches. Secondly, the spatial organisation of a CNN stays largely intact (with a certain amount of receptive field

[2]Using caffe's official weights: `https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet`.

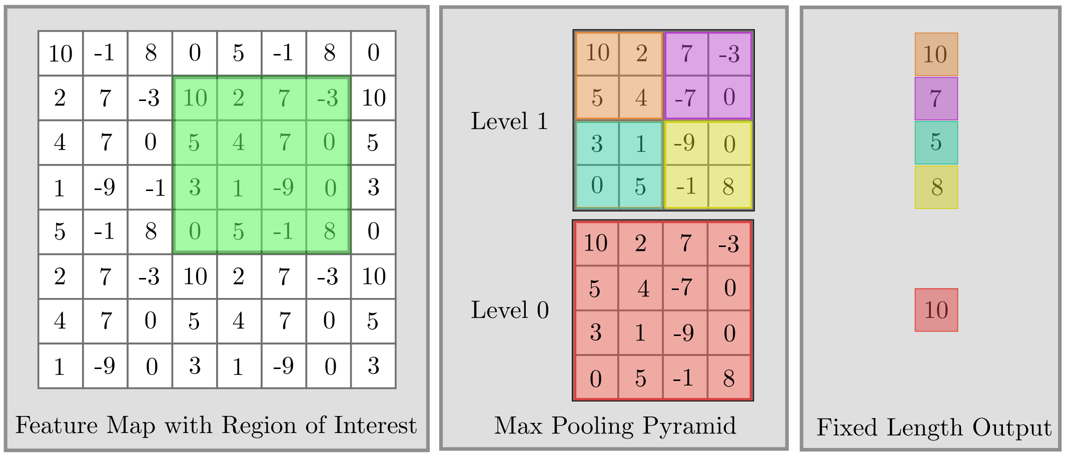


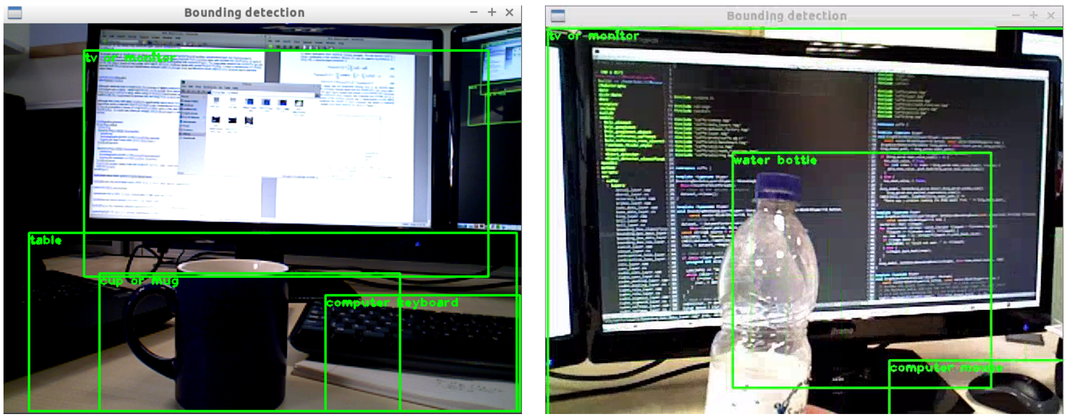
Figure 6.4: Spatial Pyramid Pooling transforms regions of interest within a feature map into a fixed length vector that can be used by a fully connected layer.

'bleeding') up to the final convolutional layer. A more efficient solution we explored was to perform a single CNN forward-pass of the whole image with a classifier CNN and use the pre-calculated feature map at a higher level to classify all of the object proposals in parallel. For classification we opted for the simpler pure-CNN approach of using the output of the Softmax layer directly and dropping the SVMs used in R-CNN.

The issue with classifying variable size region proposals based on higher-level feature maps is that the final fully connected layer requires a constant size of input. The solution proposed by [He et al., 2014] was a final Spatial Pyramid Pooling (SPP) layer, which produces a constant sized output regardless of the input size, using a pyramid of increasingly fine max-pooling grids (see Figure 6.4). To greatly speed up classification of object proposals, a CUDA-based SPP [He et al., 2014] layer was implemented in *Caffe*. In [He et al., 2014] they showed that SPP could be used to greatly speed up the processing of regions, however they continued to use Selective Search for the region proposals.

In our experiments the SPP layer was designed to directly accept region proposals in CUDA from the MultiBox CNN. To allow video feed to be classified in real-time only the top 20 object proposals were selected to be passed to the classifier CNN. A forward-pass of a frame through both the proposal and classification CNNs takes only 52ms on a GTX Titan Black GPU, allowing objects to be detected live in 20Hz video as shown in Figure 6.5.

For the classification network another GoogLeNet CNN was fine-tuned to operate on the 200 object classes in the ILSVRC-2014 object detection challenge using the standard challenge training set. We evaluated the mean Average Precision (mAP) of the various explored object proposal methods using the same SPP classifier on the ILSVRC-2014 object detection validation set [Russakovsky et al., 2015].[3] Using ground truth region proposals the system achieved 52.4% mAP. Not only was MultiBox real-time capable but it also proved to be the best proposal system of those evaluated, achieving 21.9% mAP. The MCG and BING proposal systems achieved 11.9% and 5.2% mAP respectively.



(a) A clear frame with correct detections. (b) An example with poor localisation.

Figure 6.5: Example frames from the designed real-time object detection pipeline.

6.3.3 The Evolution To Mask R-CNN

Shortly after initial work on this project was completed a number of approaches aiming to speed up object detection methods were published. These approaches developed new techniques that have been incorporated into many subsequent object detection systems.

In Fast R-CNN [Girshick, 2015] the detection portion of the pipeline (i.e. excluding the proposal time itself) was sped up to run at 300ms on 2,000 region proposals. The speed up in the detection system was attained in the same manner as the system described above. The whole image is first processed with a set of convolutional layers and then SPP is used on regions of interest on the calculated feature map. For SPP they chose a single-level pyramid which became known as an 'RoIPool'

[3] http://www.image-net.org/challenges/LSVRC/2014/

layer. They also dropped the use of the SVM and instead use the Softmax classifier which they experimentally showed outperformed the SVM approach in Fast R-CNN. Unlike our method described above, proposals for Fast R-CNN were still produced via Selective Search and so the system could not be used at real-time frame-rates. Although the initial proposals were not entirely deep learned as above, the classifier CNN itself was trained with a multi-task loss in order to refine Selective Search's region proposals by producing offsets for each object class. In our work above, the classifier network simply took the detected regions as a given.

[Ren et al., 2015] produced Faster R-CNN a system which could operate at 5Hz including both region proposals and classification. To achieve this performance they used the Fast R-CNN method for detection but as done above also used a Deep Learned method for region proposals, called the Region Proposal Network (RPN). Unlike our method above, they unify the detection and classification approaches to an even greater extent by sharing the same convolutional layer computation for both tasks and trained in an alternating manner. Additionally they did not follow the MultiBox method and instead used a Fully Convolutional approach. For each convolutional stride of the feature map they produce k object proposals and object confidence scores. The object proposals are centred on the image location of the convolution, but are parametrised with offsets relative to k anchor boxes of multiple scales and aspect ratios. This technique is translation-invariant unlike the MultiBox approach which defines anchors in terms of absolute image coordinates.

The 'YOLO' network [Redmon et al., 2016] was one of the first published Deep Learned object detectors to operate at (and beyond) real-time frame rates ($\approx$45Hz). This was achieved by completely separating the classification and bounding box regression tasks. The image is divided into a uniform grid of cells and at each grid location a bounding box is regressed, object confidence is predicted, and the class of that grid cell is predicted independently and in parallel. This approach is in contrast to the 'first detect then classify' system stemming from R-CNN as the classifier in YOLO is based only on the grid square rather than on a complete region defined by the regressed bounding box.

Mask R-CNN [He et al., 2017] consists of a straightforward addition to the Faster R-CNN system and also can operate at $\approx$5Hz. As well as producing a classification for the output of the RPN they produce in parallel for each object detection a binary instance mask segmenting the object itself. It was found to be important to prevent competition between classes by predicting a separate mask output for each of the classes. In this way the classification of the object does not depend on the instance segmentation; instead the classification is used to select which of the predicted instance masks are output as the final segmentation. This binary mask and semantic classification are used as input to our Fusion++ system. They also modified the RoIPool method to reduce the quantization error of sampling feature maps by using bilinear interpolation, which they call RoIAlign. The improved alignment was found to be particularly important for the system's performance.

6.4 Method

The complete Fusion++ system is visualised in Figure 6.6. From RGB-D input, a coarse background TSDF is initialised for local tracking and occlusion handling (Section 6.4.3). If the pose changes sufficiently or the system appears lost, relocalisation (Section 6.4.4) and graph optimisation (Section 6.4.5) are performed to arrive at a new camera location, and the coarse TSDF is reset. In a separate thread RGB frames are processed by Mask R-CNN and the detections are filtered and matched to the existing map (Section 6.4.2). When no match occurs, new TSDF object instances are created, sized, and added to the map for local tracking, global graph optimisation, and relocalisation. On subsequent frames, associated foreground detections are fused into the object's 3D 'foreground' mask alongside semantic and existence probabilities (Section 6.4.1).

6.4.1 TSDF Object Instances

Our map is composed of object instances reconstructed within separate TSDFs, $\mathcal{V}^o$. The TSDF pose, $\mathbf{T}_{WO} \in SE(3)$, is defined relative to the object coordinate frame, $\underrightarrow{\mathcal{F}}_O$, which has its origin at the centre of the volume and is axis-aligned with the voxels.

Initialisation and resizing: Detections not matched by the procedure described in 6.4.2 are used to initialise an appropriately sized and positioned instance TSDF. In the k^{th} frame each detection i produces a binary mask, M_i^k. We project all of the masked image coordinates, $\mathbf{u} = (u_1, u_2)$, into $\underrightarrow{\mathcal{F}}_W$ using the depth map, $D_k(\mathbf{u})$, and estimated camera pose, $\tilde{\mathbf{T}}_{WC}^k$ (see Equation 2.19 in Chapter 2 for details).

To robustly size the TSDF in the presence of masks which can occasionally include

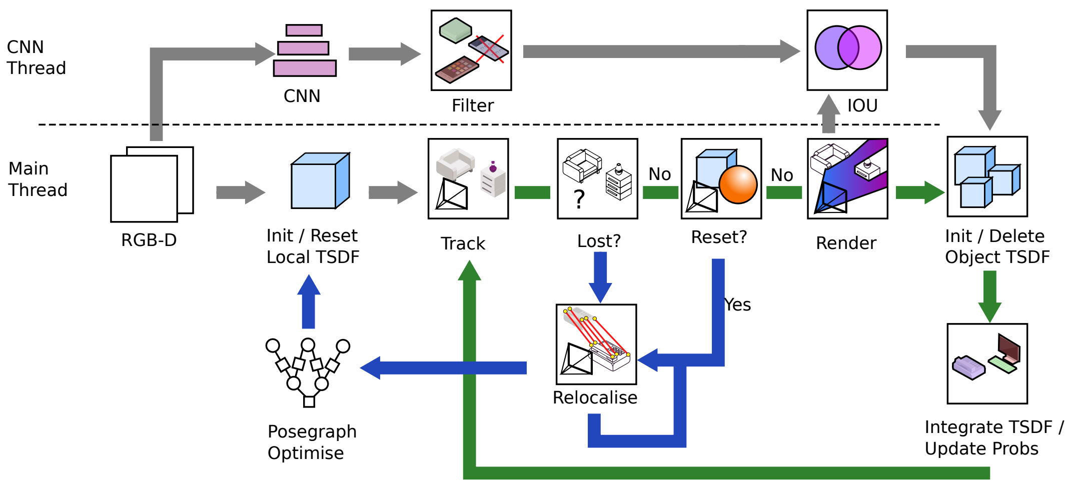


Figure 6.6: An overview of the Fusion++ operating loop. RGB-D input data is processed in two separate threads: **(top)** the CNN thread detects object instances and attempts to match them to existing instances in the map. **(bottom)** the main thread performs tracking, TSDF initialisation/integration, and relocalisation/posepraph optimisation. The SLAM system tracking loop (**green arrows**) continues either until the camera exits the background TSDF volume or tracking fails. When either occur the system enters relocalisation mode (**blue arrows**) and attempts to relocalise itself against historic snapshots. Once relocalised, the camera and object instance pose-graph is optimised and the background TSDF volume is reset and centered around the current camera pose before returning to the tracking loop.

far-away background surfaces, we do not directly accept the maximum and minimum of this point cloud. Instead we use the 10^{th} and 90^{th} percentiles of this point cloud (separately for each axis) to define points $\mathbf{p}_{10}$ and $\mathbf{p}_{90}$ respectively, which are used to calculate the volume centre, $\mathbf{p}_o = \frac{\mathbf{p}_{90}+\mathbf{p}_{10}}{2}$, and volume size, $s_o = m\|(\mathbf{p}_{90} - \mathbf{p}_{10})\|_\infty$. We use an m of 1.5 to account for erosion and provide additional padding.

Each instance TSDF has an initial fixed resolution of r_o^3, which we choose to be 64^3. The total physical size of the volume, s_o, is used to calculate the physical size of a single voxel, $v_o = \frac{s_o}{r_o}$. This system means that small objects will be reconstructed with fine details and large objects more coarsely, making the map as useful as possible for a given memory footprint.

During operation matched objects may need to be re-sized as new detections include additional areas. To do this, the point cloud of the current mask described above is combined with a similarly eroded point cloud generated from the current TSDF reconstruction. The 3D volume encompassing them both is used to calculate the new volume centre and size as before. Early attempts which sampled the TSDF with a general transformation quickly led to severe deterioration of the reconstruction as a result of aliasing. Therefore when re-sizing we quantize the transformation by only translating by discrete multiples of v_o. If the size of the volume must increase, we maintain the voxel's physical size, v_o, but increase r_o. We also ensure that r_o maintains an even parity:

$$r_o = \begin{cases} \lfloor \frac{s_o}{v_o} \rfloor + 1, & \text{if} \lfloor \frac{s_o}{v_o} \rfloor \text{is odd} \\ \lfloor \frac{s_o}{v_o} \rfloor, & \text{otherwise,} \end{cases} \tag{6.4}$$

and limit the maximum voxel resolution to 128, by re-initialising the volume as though new if $r_o > 128$. Finally we limit the maximum object size to be 3m.

To initialise an instance we require that the volume centre be within 5m of the camera to prevent noise at far depth ranges from producing poorly reconstructed objects. As it is possible to produce multiple 3D boxes in the same volume area we use a 3D version of the NMS algorithm and require that the 3D axis-aligned bounding box IoU with any other volume in the map is less than 0.5. When an object centre is moved, the pose-graph node and associated measurements are also updated as described in Section 6.4.5.

Integration: For integrating surface measurements from a depth map, D^k, into a volume, $\mathcal{V}^o$, we follow the KinectFusion approach of [Newcombe et al., 2011a].[4] The depth map here is assumed to be of the kind produced from a Kinect device, consisting of measurements of the z-axis component of each pixel along the camera's principal axis (see Figure 2.3, Chapter 2). $\mathcal{V}^o$ stores at each discrete voxel location, $\mathbf{v}_o = (v_x, v_y, v_z)$, both the current normalised truncated signed distance value, $S^o_{k-1}(\mathbf{v}_o)$, and its associated weight, $W^o_{k-1}(\mathbf{v}_o)$. For visualisation purposes we also store and integrate RGB values in each voxel in a manner identical to the SDF values which produces a coarse texture for each object.

Discrete voxel locations can be mapped to their continuous spatial location ${}_O\mathbf{p}(\mathbf{v})$ in $\underrightarrow{\mathcal{F}}_O$, and any continuous spatial location within the volume can be mapped to the voxel within which it resides. Each $\mathbf{v}_o$ is transformed into the k^{th} camera frame using the camera pose estimate, $\tilde{\mathbf{T}}^k_{CW}$, and known object pose, $\mathbf{T}_{WO}$:

$${}_C\mathbf{p}(\mathbf{v}_o) = \tilde{\mathbf{T}}^k_{CW} \mathbf{T}_{WO} {}_O\mathbf{p}(\mathbf{v}_o). \tag{6.5}$$

[4] The implementation is based on https://github.com/GerhardR/kfusion.

${}_C\mathbf{p}(\mathbf{v}_o)$ can then be projected to an image coordinate, $\mathbf{u}_{\mathbf{v}_o}$, using the camera's intrinsic matrix (Equation 2.18, Chapter 2). The Signed Distance Function (SDF) represents the distance of a given voxel from the surface described by the depth measurement. We use the convention of positive SDF values outside the surface and negative values within the interior of the surface. The SDF is calculated as:

$$S^o(\mathbf{v}_o) = (D^k(\mathbf{u}_{\mathbf{v}_o}) - [{}_C\mathbf{p}(\mathbf{v}_o)]_3)\frac{\|{}_C\mathbf{p}(\mathbf{v}_o)\|}{[{}_C\mathbf{p}(\mathbf{v}_o)]_3}, \tag{6.6}$$

where $[\mathbf{p}]_3$ indicates the z-component of $\mathbf{p}$ corresponding to the planar depth image, D. We do not know how thick the object is, and so values that are deep within the object ($S^o(\mathbf{v}_o) > -\mu$) are considered occluded and not updated. Here we choose the threshold, μ, based on the physical size of the voxels $\mu = 4v_o$. $S^o_{k-1}(\mathbf{v}_o)$ and $W^o_{k-1}(\mathbf{v}_o)$ are then updated via weighted averaging using the normalised measurement, $\frac{S^o(\mathbf{v})}{\mu}$, truncated to be in the range $[-1, 1]$ which in physical distance is $[-\mu, \mu]$:

$$S^o_k(\mathbf{v}) = \max(-1, \min(1, \frac{S^o_{k-1}(\mathbf{v})W^o_{k-1}(\mathbf{v}) + \min(\frac{S^o(\mathbf{v})}{\mu}, 1)}{W^o_{k-1}(\mathbf{v}) + 1})). \tag{6.7}$$

The weighting scheme follows simple addition for the first 100 measurements and then is capped to prevent voxel weights becoming overly rigid after many updates:

$$W^o_k(\mathbf{v}) = \min(W^o_{k-1}(\mathbf{v}) + 1, 100). \tag{6.8}$$

This integration step is performed in our system on every frame where the TSDF volume is visible, when 50% of TSDF pixels are validly tracked and the ICP RMSE is less than 3cm (these error metrics are described in more detail in Section 6.4.3). This is to maintain the reconstruction quality of instances when the camera pose may have drifted relative to them.

It is also important to note that the above surface integration is performed throughout the entire volume, regardless of whether it is a masked instance region or not. The TSDF's zero-crossing point already correctly defines the surface/free-space boundary for all of the geometry within the volume, so if just one instance were to exist within the volume (surrounded only by free-space) then the volume's reconstruction would properly segment its instance. In practice an object's volume almost always includes geometry from other instances as well, such as its supporting surface and other objects in close proximity. This makes it necessary to also encode which of the voxels within the volume are associated to that volume's instance 'foreground.'

To store which voxels correspond to this instance's foreground we fuse the binary instance mask detections into the voxel's state. We view each positive or negative detection as the result of a binomial trial sampled from a latent foreground probability, $p^o(\mathbf{v}_o \in \text{foreground})$. We store foreground, $F^o_{k-1}(\mathbf{v}_o)$, and not foreground, $N^o_{k-1}(\mathbf{v}_o)$, detection counts as the (α, β) shape parameters in a Beta distribution conjugate prior which are initialised with $(1, 1)$. When a new detection is matched and $S^o(\mathbf{v}_o) > -\mu$ then we also update the detection counts using the corresponding instance mask, M^i_k,

$$F^o_k(\mathbf{v}_o) = F^o_{k-1}(\mathbf{v}_o) + M^i_k(\mathbf{u}_{\mathbf{v}_o}), \tag{6.9}$$

with the 'not foreground' detection counts simply being the inverse,

$$N^o_k(\mathbf{v}_o) = N^o_{k-1}(\mathbf{v}_o) + (1 - M^i_k(\mathbf{u}_{\mathbf{v}_o})). \tag{6.10}$$

Finally we compute whether a voxel is part of the foreground by calculating the expectation from the beta distribution,

$$E[p^o(\mathbf{v}_o)] = \frac{F^o_{k-1}(\mathbf{v}_o)}{F^o_{k-1}(\mathbf{v}_o) + N^o_{k-1}(\mathbf{v}_o)}, \tag{6.11}$$

and use a decision threshold of $E[p^o(\mathbf{v}_o)] > 0.5$ for rendering.

A heatmap visualisation of the expectation this produces at the surface crossing point is shown in Figure 6.7. However it is important to note that the heatmap exists everywhere in the volume in a manner analogous to the SDF itself rather than just at the surface crossing illustrated here. Figure 6.7 also neatly illustrates that it is possible for the same surface element to be reconstructed multiple times in separate TSDFs; a part of each bag is also reconstructed in the TSDF for the other but that part is considered background within the TSDF.

Raycasting: For tracking, data association, and visualisation we render depth, normals, vertices, RGB, and object indices using raycasting. Within each object volume $\mathcal{V}^o$ we step along the ray with a stepsize of v_o (and $0.5v_o$ when $S^o_k(\mathbf{v}_o) < 0.8$, where $S^o_k(\mathbf{v}_o)$ is the SDF normalised by μ) and search for the zero-crossing point where $E[p^o(\mathbf{v})] > 0.5$. Both $S^o_k(\mathbf{v}_o)$ and $E[p^o(\mathbf{v})]$ are trilinearly interpolated from neighbouring voxels to smooth the representation. We store the ray length of the nearest of these intersections to avoid searching through another object volume which starts beyond the current hit point.

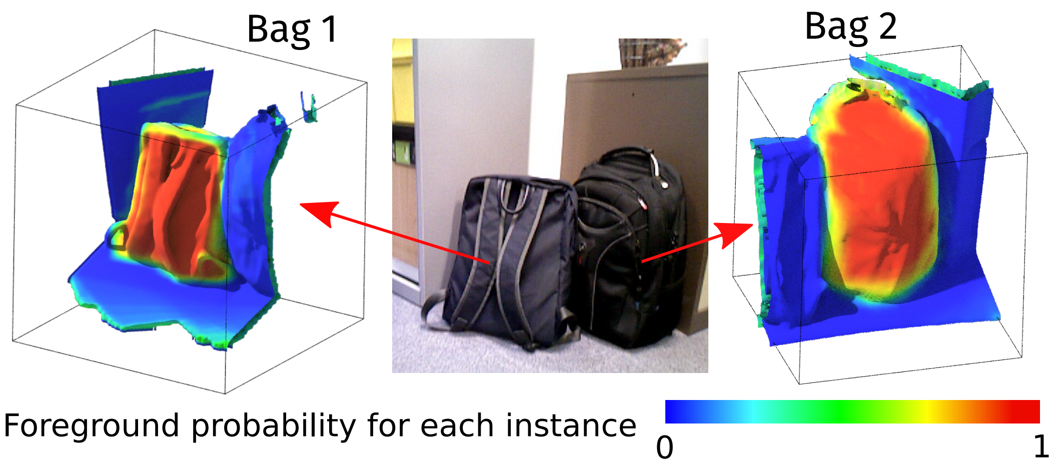


Figure 6.7: A heatmap of the volumetric foreground probability on the implicit surface of two nearby objects. It can be seen that, although portions of each bag are reconstructed in each others volumes, the foreground probability heatmap allows each instance to be neatly segmented within its own volume while the other instance can be ignored as background. If the foreground probability falls below 0.5 it is not rendered for either tracking or data association, but it is shown here for illustration purposes.

Occluding surfaces that are not objects would not be respected if the representation stored objects alone. If the background TSDF described in Section 6.4.3 is available and either no intersection with a foreground object occurs or the intersection is farther than 5cm behind the background TSDF intersection, then the background TSDF ray intersection is used instead. The 5cm threshold is there to give priority to the reconstructed objects. If a background TSDF is not available a depth consistency check against the raw depth image can also be used to properly occlude objects.

Existence Probability: To prevent spurious instances from building up over time, we also model the probability of each instance's existence as $p(o)$ using the Beta distribution, in a manner identical to the foreground mask. For any frame where a predicted instance should be clearly visible (i.e. our raycasted image has more than 50^2 pixels of that instance which is not occluded by other scene elements), then if the instance has been associated to a detection its existence count, e_o, is incremented, and if not its non-existence count, d_o, is incremented. If $E[p(o)]$ falls below 0.1, the instance is deleted and the object node with all associated edges are removed from the pose graph (described in Section 6.4.5).

Semantic Labels: Each TSDF also stores a probability distribution over potential class labels, c. Mask R-CNN provides a probability distribution $\tilde{p}(c_k|I_k)$ over the classes given the image, I_k. We found that the standard multiplicative Bayesian update scheme used in SemanticFusion (described in Equation 4.10 of Chapter 4) often leads to an overly confident class probability distribution, with scores unsuitable for ranking in terms of object detection. As described in Section 6.4.2 we only accept detections with a class probability over 50%. Over time the independence assumption in the Bayesian update leads to object probabilities clustered close to 100%, as the class distribution between views is, in practice, closely related. To avoid this, instead of treating these predictions as independent probabilities, we view them as 'noisy' measurements of a latent class distribution, and so fuse multiple associated detections via simple averaging:

$$p(c_k|I_{0..k}) = \frac{1}{k}\sum_{i=1}^{k}\tilde{p}(c_i|I_i). \tag{6.12}$$

Intuitively, we want repeated identical class distribution predictions to result in that same distribution, rather than to compound to an almost certainty by assuming independence. We found that this approach produced a more even class probability distribution in practice than the Bayesian update scheme.

6.4.2 Detection and Data Association

Detections from the Mask R-CNN model [He et al., 2017] for a given frame k contain instances i with a binary mask M_k^i and class probability distribution $p(c_k^i|I_k)$. A forward pass takes ~250ms, and although our system is not real-time, this still represents a significant bottleneck and so is performed in a parallel thread. For GPU memory efficiency, we take only the top 100 detections scored according to the Region Proposal Network's 'object' score.

Ideally we wish to correctly size the TSDF when we first initialise it for a given object. In the context of long-term use by an active agent a method to achieve this is to wait until the object is viewed from a favourable pose where it is reasonably close to the viewer and is centred in the image. We also found that the Mask R-CNN predictions deteriorated when objects were only partially visible near the border of the image. For both of these reasons we filter for masks not near the image border (within 20 pixels) and where both $\max(p(c_k^i|I_k)) > 0.5$ and $\sum M_k^i > 50^2$. Over shorter time-horizons this approach comes at the cost of missing large objects which are only occasionally viewed at the correct distance to be both entirely visible and not too small.

After local tracking (Section 6.4.3) we use the estimated camera pose and TSDFs already initialised in the map to raycast a binary mask M_k^o for object instances o in the current view. We map each detection i to a single instance o by calculating the intersection of the two as a proportion of the *detection's* area:

$$a_{\text{detect}}(i, o) = \frac{\sum M_k^o \cap M_k^i}{\sum M_k^i}, \tag{6.13}$$

and assigning the detection to the largest intersection, $\tilde{o} = \operatorname{argmax}_o a_{\text{detect}}(i, o)$, where $a_{\text{detect}}(i, \tilde{o}) > 0.2$, otherwise the detection is unassigned. For the integration step, each detection that has been mapped to the same instance is combined by taking the union of the detection masks, and the average of the class probabilities.

6.4.3 Layered Local Tracking

We maintain an instance-agnostic coarse background TSDF, a, to assist local frame-to-model tracking where/when there are no instances and to handle occlusions. It has a resolution of 256^3 with a voxel size of 2cm. Its initialisation point, ${}_W\mathbf{p}_a = \mathbf{T}_{WC}^k[0 \quad 0 \quad 2.56]^\intercal$, is 2.56m along the z-axis in the camera frame, $\underrightarrow{\mathcal{F}}_C$, to prevent wasted volume as in [Whelan et al., 2015a]. The volume is reset when its new initialisation point exits a spherical threshold (1.28m) around the previous volume centre, *i.e.* $\|{}_W\mathbf{p}_a - \mathbf{T}_{WC}^k[0 \quad 0 \quad 2.56]^\intercal\|_2 > 1.28$.

We combine the background TSDF with individual instances to raycast (Section 6.4.1) a 'layered' reference frame, denoted r, with vertex map, V_r, normal map, N_r, and object index map, X_r, from the previous camera pose, $\mathbf{T}_{WC_r}$, with vertices and normals defined in the world frame, $\underrightarrow{\mathcal{F}}_W$. The transform to the live frame, denoted l, is estimated by aligning the live depth map (after bilateral filtering) to the rendered maps with ICP using projective data association and a point-to-plane error as described in detail in Section 2.4.1 of Chapter 2.

The valid vertex set, V_{valid}, used for tracking includes any $\mathbf{u}_l$ with a corresponding vertex and normal, where there is a corresponding $\mathbf{u}_r$ with a valid vertex and normal, and where $N_r(\mathbf{u}_r) \cdot N_l(\mathbf{u}_l) < 0.8$ and $\|V_r(\mathbf{u}_r) - \tilde{\mathbf{T}}_{WC_l} V_l(\mathbf{u}_l)\|_2 < 0.1$m. We minimize this NLLS problem using the Gauss-Newton algorithm. The 6×6 Hessian approximation, $\mathbf{J}_{\text{icp}}^\intercal \mathbf{J}_{\text{icp}}$, and 6×1 error Jacobian, $\mathbf{J}_{\text{icp}}^\intercal \mathbf{r}_{\text{icp}}$, are reduced in parallel on the GPU and solved on the CPU using SVD and back substitution. We use a three-level coarse-to-fine pyramid scheme with 5 Gauss-Newton iterations per level.

We perform an additional reduction on the GPU to produce the same system of equations partitioned into pixels, $\mathbf{u}_l$, associated to each instance in $X_r(\mathbf{u}_r)$ for pose-graph optimisation and to produce per-instance error metrics. The error metrics are the ICP RMSE, $(|V_{\text{valid}}|^{-1} E_{\text{icp}}(\tilde{\mathbf{T}}_{WC_l}))^{\frac{1}{2}}$, and the proportion of validly tracked pixels, $\frac{|V_{\text{valid}}|}{|V_l|}$. These are used for instance integration and to check whether local tracking is lost. We consider local tracking to be lost when the total ICP RMSE is greater than 0.05m or when at least 10% of the image consists of instance TSDFs and less than half of the pixels are validly tracked, in which case we enter relocalisation mode as described below.

6.4.4 Relocalisation

If the system is lost or we reset the coarse TSDF, we perform relocalisation to align the current frame to the current set of instances (if there are any). We found that direct dense ICP methods using only the volume reconstructions did not produce accurate results for wide baseline relocalisation as they are sensitive to the initial pose and small objects were often ambiguous without texture constraints. Although alternative dense methods may also prove useful here, we took the approach of using snapshots of sparse BRISK features[5] (with a detection threshold of 10) projected to 3D using the depth map. For a given detection of an object, if there is no existing snapshot of the object within 15° view angle difference, then we add a new snapshot of the object from that pose (see Figure 6.8).

To relocalise we perform 3D-3D RANSAC against each instance where the dot product with the predicted class distribution is greater than 0.6. We use the OpenGV library as our back-end [Kneip and Furgale, 2014] with a minimum of 5 inlier features (within 2cm) to match each object individually. If we find one or more matching objects in the scene, we run a final 3D-3D RANSAC on every point in the scene (from all objects and the background jointly) with a minimum of 50 inlier features (within 5cm) to arrive at a final camera pose.

The calculated pose is the only output of the relocalisation system. To add new constraints to the pose-graph the pose is used to render a new reference image of

[5]BRISK v.2 with homogeneous Harris scale space corner detection on only the highest image resolution.

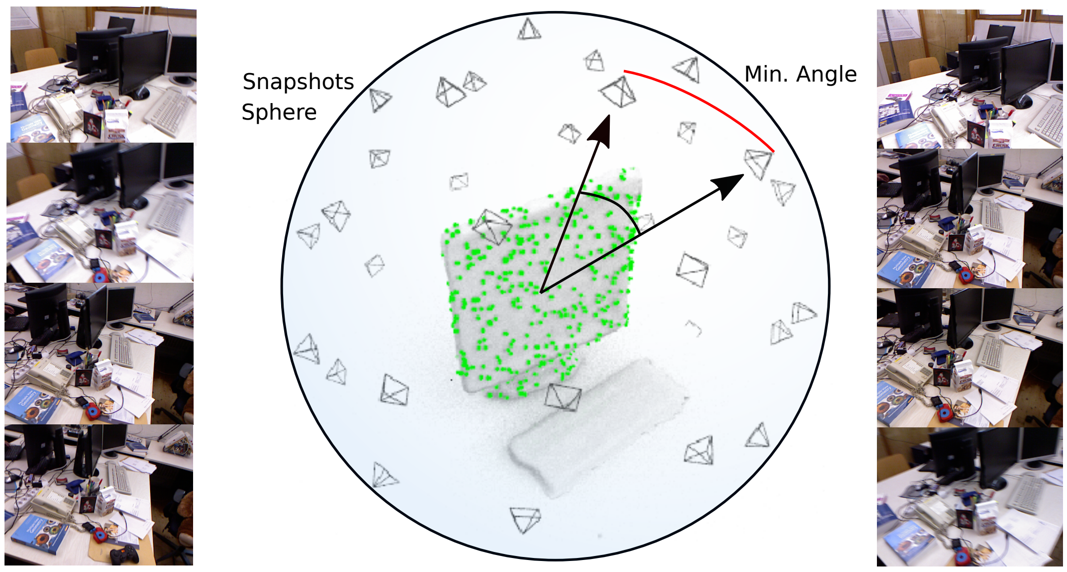


Figure 6.8: Re-localisation snapshots around an instance. For each object we add a new snapshot if there is no existing snapshot of the object within a 15° view angle difference.

the map. ICP is then performed on the reference image to produce the partitioned measurements and constraints required for pose graph optimisation as described below.

6.4.5 Object-Level Pose Graph

Our pose-graph formulation is similar to that of [Salas-Moreno et al., 2013]. For every frame with a Mask R-CNN detection (including coarse TSDF resets), we add a new camera pose node to our graph. When a new instance, index o, is initialised, a corresponding landmark node is added to the graph, defined by the coordinate frame attached to the centre of the object's volume, $\mathbf{p}_o$. The first camera pose node is fixed and defined to be the origin of the world frame, $\underrightarrow{\mathcal{F}}_W$. Each node consists of a full $SE(3)$ transformation from object to world, $\mathbf{T}_{WO}$, or camera to world, $\mathbf{T}_{WC}$, and the measurements are $SE(3)$ relative pose constraints between nodes.

Unfortunately the geometry of many of the discovered objects and the noisy partial reconstructions during online operation meant that ICP on individual objects was frequently unsuccessful. To produce the edge measurements for each object, we instead used the ICP error terms calculated from the combined layered tracking reference image, but with errors partitioned to correspond to the pixels of the specific object o (for object-camera constraints), or the instance-agnostic background a (for camera-camera constraints). To ensure that the measurement coincides with the minimum of the partitioned set's quadratically approximated error function, an additional Gauss-Newton step is performed. The step uses the partitioned $\mathbf{J}^{o}_{\text{icp}}$ (see Section 6.4.3) to produce 'virtual' relative pose measurements, $\tilde{\mathbf{T}}'^{a}_{C_{k-1}C_k}$, between camera nodes, and $\tilde{\mathbf{T}}'^{o}_{OC_k}$, between camera and landmark objects. The resulting measurement errors for the graph factors are:

$$\mathbf{e}_{\text{cc}}(\mathbf{T}_{C_{k-1}W}, \mathbf{T}_{WC_k}) = \log((\tilde{\mathbf{T}}'^{a}_{C_{k-1}C_k})^{-1}\mathbf{T}_{C_{k-1}W}\mathbf{T}_{WC_k}), \tag{6.14}$$

$$\mathbf{e}_{\text{oc}}(\mathbf{T}^{o}_{OW}, \mathbf{T}_{WC_k}) = \log((\tilde{\mathbf{T}}'^{o}_{OC_k})^{-1}\mathbf{T}^{o}_{OW}\mathbf{T}_{WC_k}). \tag{6.15}$$

For every relative measurement, we approximate the inverse measurement covariance by $\mathbf{\Sigma}^{-1} = \mathbf{J}^{o\intercal}_{\text{icp}}\mathbf{J}^{o}_{\text{icp}}$. This is an approach similar to that of [Bengtsson and Baerveldt, 2003] which was for 2D scan-matching except that it neglects the scalar relating to the unbiased estimate of the assumed Gaussian noise. An alternative approach proposed by [Censi, 2007] also exists and could in future be explored.

The way perturbations are modelled differs between the ICP algorithm and the employed pose graph optimiser, so we need to transform the covariance by considering the relation between the local perturbations. The graph optimiser models perturbations to relative pose measurements, $\boldsymbol{\zeta}_{\text{pg}}$, via $\tilde{\mathbf{T}}'^{o}_{O'C_k} = \tilde{\mathbf{T}}'^{o}_{OC_k}\exp(\boldsymbol{\zeta}_{\text{pg}})$ (equivalently for $\tilde{\mathbf{T}}'^{a}_{C_{k-1}C_k}$). To ensure our information matrix properly corresponds to perturbations $\boldsymbol{\zeta}_{\text{pg}}$, it is necessary to convert $\mathbf{J}_{\text{icp}}$. As can be seen in Equation 2.42 (Chapter 2), $\mathbf{J}_{\text{icp}}$ is with respect to perturbations applied via $\tilde{\mathbf{T}}_{W'C_k} = \exp(\boldsymbol{\zeta}_{\text{icp}})\tilde{\mathbf{T}}_{WC_k}$. The relation between $\boldsymbol{\zeta}_{\text{icp}}$ and $\boldsymbol{\zeta}_{\text{pg}}$ is:

$$\exp(\boldsymbol{\zeta}_{\text{icp}})\mathbf{T}_{WC_k} = \mathbf{T}^{o}_{WO}\tilde{\mathbf{T}}'^{o}_{OC_k}\exp(\boldsymbol{\zeta}_{\text{pg}}), \tag{6.16}$$

$$\boldsymbol{\zeta}_{\text{icp}} = \log(\mathbf{T}_{WC_k}\exp(\boldsymbol{\zeta}_{\text{pg}})\mathbf{T}^{-1}_{WC_k}) = \text{Adj}_{\mathbf{T}_{WC_k}}\boldsymbol{\zeta}_{\text{pg}}, \tag{6.17}$$

$$\mathbf{J}_{\text{pg}} = \frac{\partial \boldsymbol{\zeta}_{\text{icp}}}{\partial \boldsymbol{\zeta}_{\text{pg}}} = \text{Adj}_{\mathbf{T}_{WC_k}}. \tag{6.18}$$

The derivation for camera nodes follows an identical pattern and results in the same transformation. The new information matrix therefore becomes:

$$\mathbf{H}_{\text{pg}} = \mathbf{J}^{\intercal}_{\text{pg}}(\mathbf{J}^{o\intercal}_{\text{icp}}\mathbf{J}^{o}_{\text{icp}})\mathbf{J}_{\text{pg}}. \tag{6.19}$$

The final error to be minimised in the pose graph is the sum over all the edges from the camera to objects, $\mathcal{O}$, and camera to camera, $\mathcal{C}$, given their state, the measurement, and the information matrix,

$$E_{\text{pg}} = \sum_{\text{cc}\in\mathcal{C}} L_\sigma(\mathbf{e}_{\text{cc}}^\intercal \mathbf{H}_{\text{pg}} \mathbf{e}_{\text{cc}}) + \sum_{\text{oc}\in\mathcal{O}} L_\sigma(\mathbf{e}_{\text{oc}}^\intercal \mathbf{H}_{\text{pg}}, \mathbf{e}_{\text{oc}}), \tag{6.20}$$

where L_σ denotes a robust Huber kernel. We solve this graph in the g2o [Kümmerle et al., 2011] framework using sparse Cholesky decomposition and Levenberg-Marquart optimisation. After optimisation we update the pose of the instance TSDFs and the camera before initialising the new coarse TSDF to that pose and continuing with local tracking.

As described in Section 6.4.1, when a landmark is re-sized, its centre, $\mathbf{p}_o$, can also be adjusted from $\underrightarrow{\mathcal{F}}_O$ to a new frame $\underrightarrow{\mathcal{F}}_{O'}$ via the transform $\mathbf{T}_{O'O}$. In this case we also transform the corresponding node variable, $\mathbf{T}^o_{WO'} = \mathbf{T}^o_{WO}\mathbf{T}^{-1}_{OO}$, as well as the measurement for every edge connected to that node, $\tilde{\mathbf{T}}'^o_{O'C} = \mathbf{T}_{O'O}\tilde{\mathbf{T}}'^o_{OC}$.

6.5 Experiments

We evaluate the performance and memory usage of our system on a Linux system with an Intel Core i7-5820K CPU at 3.30GHz, and an NVIDIA GeForce GTX1080 Ti GPU with 11.175GB of memory. Our core pipeline is implemented in Python and uses Tensorflow for instance predictions, and Python wrappers around other core components which are developed in C++ and/or CUDA, such as KFusion, g2o, BRISK, and OpenGV. Our input is standard VGA (640×480) resolution RGB-D video. To allow for reproducibility, instead of running an asynchronous CNN thread we here perform predictions synchronously every 30 frames.

Our Mask-RCNN uses the ResNet-101 base model [He et al., 2016] (up to the final layer in the conv4_x block) and is finetuned from the publicly available weights and implementation [Wu et al., 2016b].[6] The base weights are pre-trained on the COCO dataset, so it is necessary to finetune on a set of classes more suitable for indoor scenes. For finetuning we use the NYUv2 training set and lock the ResNet-101 backbone weights from the COCO pretraining. As the COCO dataset consists of 80 classes we resize and reinitialise the class-specific upper layers of Mask R-CNN

[6] `models.tensorpack.com`

and Faster R-CNN to the 40 class split defined by [Gupta et al., 2013]. We train using SGD with momentum of 0.9 for 30 epochs and a learning rate of 0.001.

As well as simply having the wrong type of classes we also found that the class-agnostic instance predictions from the base COCO weights were worse than that of the fine-tuned CNN. We believe this is largely due to the difference between image domains of the COCO dataset and that of a cluttered indoor scene typified by the NYUv2 dataset. We converted the NYUv2 test set into the required format to be used as part of the COCO Evaluation API.[7] The COCO class-agnostic Average Precision (AP) for the COCO trained weights was 8.9% vs. 19.8% for the fine-tuned weights. Figure 6.9 shows a qualitative comparison of the instance predictions.

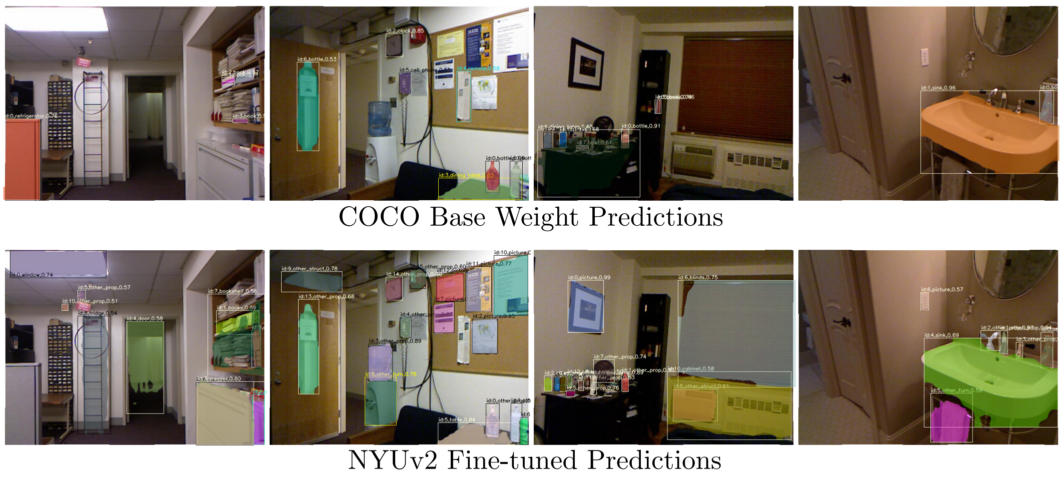


Figure 6.9: Qualitative comparison of the Mask R-CNN instance predictions on the NYUv2 test using the base COCO trained weights (top row) and predictions after fine-tuning on the NYUv2 training set (bottom row). Predictions with confidence > 0.5 are shown. Fine-tuning improves the recall of smaller objects.

6.5.1 Loop Closure and Map Consistency

To evaluate the performance of our system while repeatedly viewing a scene of instances we captured a 3,685 frame sequence of an indoor office scene. We tailored this sequence to evaluate the consistency of our map in the presence of poorly constrained geometry. For a small segment of the first loop we aim directly at the floor so that the camera pose is poorly constrained with motion parallel to the plane. After this segment we loop over the same scene for a second time to validate the loop closure approach.

[7]The COCO evaluation tools are available at: https://github.com/cocodataset/cocoapi.

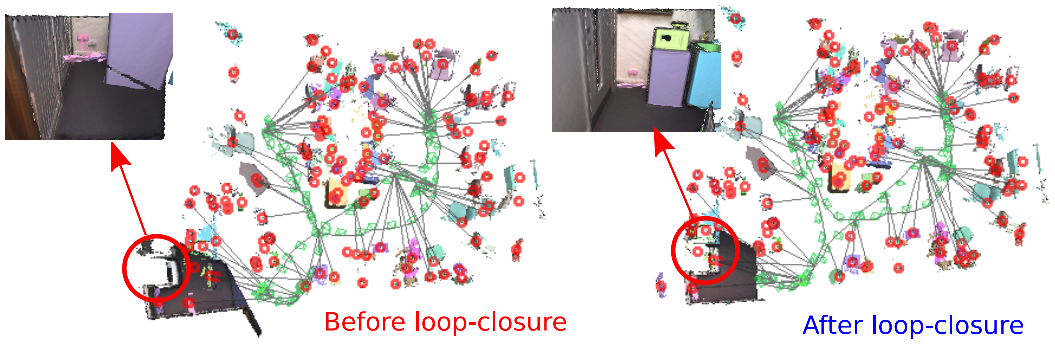


Figure 6.10: Comparison of office sequence trajectory before loop-closure (left) and after loop-closure (right). After loop-closure the object TSDF poses have been updated to align with current measurements, allowing their reconstructions to be maintained on repeated loops.

The pose-graph and loop closure view inset is shown in Figure 6.10, it can be seen that despite the accumulated drift, the system relocalises and corrects the pose graph, this allows the previously reconstructed objects to be correctly associated in future frames. Overall in the trajectory our system reconstructed approximately 100 landmark object instances. However, it must be noted that despite our filtering mechanisms, a build up of noisy partially reconstructed objects still occurs.

6.5.2 Reconstruction Quality

To evaluate the reconstruction quality we use objects from the YCB dataset which provides ground truth models [Calli et al., 2015] and reconstructs discovered objects from `sequence_0001` of the public YCB video dataset [Xiang et al., 2017]. Figure 6.11 shows a qualitative comparison against the ground truth. The missing portion of the cracker box was caused by an occlusion by another object, and a missed foreground detection on one of the few frames where the cracker box was unoccluded.

6.5.3 RGB-D SLAM Benchmark

We evaluate the trajectory error of our system against the baseline approach of simple coarse TSDF odometry, *i.e.* using the same coarse resetting background without instances layered on top, and without loop-closure pose graph optimisation. Table 6.1 shows the results. It can be seen that in all but one of the sequences evaluated, our Fusion++ system improved upon the baseline approach (while provid-

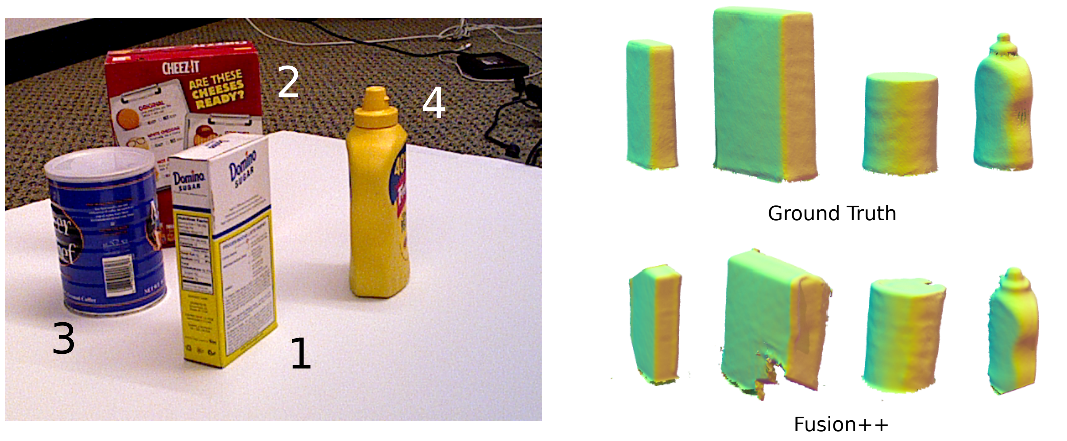


Figure 6.11: Reconstruction quality vs ground truth from `sequence_0001` of the public YCB video dataset [Xiang et al., 2017].

ing an inventory of objects as Figure 6.1 visualises for the fr2_desk sequence).

Our system does not achieve trajectory accuracies that are competitive with state-of-the-art SLAM systems [Whelan et al., 2015b, Mur-Artal and Tardós, 2017], and it would require additional work to do so, such as including joint depth and photometric tracking. However it is worth noting that the RGB-D SLAM sequences themselves are not particularly well suited for evaluating some of the strengths of the system, as they comprise of relatively short video snippets often with a single loop of a scene. Associating objects over multiple loops and an increasingly refined global map structure over longer periods of time are useful attributes for long term navigation by a robotic agent, and this is one of the goals of the present system. It was for this reason that we created the sequence with a repeated loop of an office scene (discussed above) in order to better approximate such an operating environment.

Table 6.1: RGB-D SLAM Benchmark ATE RMSE (m).

Sequence	**TSDF Odometry**	**Fusion++**
fr1_desk	0.066	**0.049**
fr1_desk2	**0.146**	0.153
fr1_room	0.305	**0.235**
fr2_desk	0.342	**0.114**
fr2_xyz	0.022	**0.020**
fr3_long_office	0.281	**0.108**

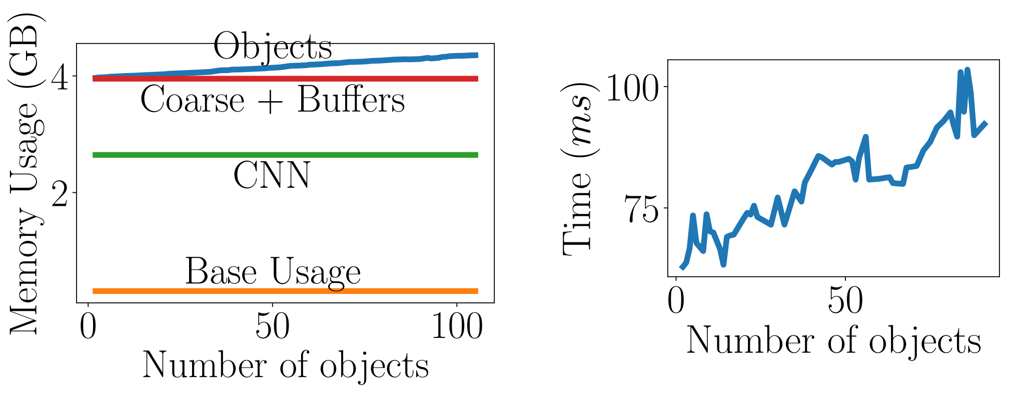


Figure 6.12: GPU memory usage and median per-frame wall clock scaling by number of objects on the office sequence. With an 11GB consumer GPU up to 2.5k object TSDF's can be stored on device. The frame rate of the system varied from 13Hz without objects to 10Hz for 100 objects due to the linear scaling of visibility checks which are coded in Python.

6.5.4 Memory and Run-time Analysis

Memory usage: We use the office sequence to evaluate the run-time performance and memory usage of our system. As memory usage scales cubically with the size of a TSDF, it is significantly more efficient to compose a map of many relatively small, highly detailed volumes in dense areas of interest than to use one large volume with a resolution equal to the smallest. After loading the CNN and image buffers, our remaining ∼7GB GPU memory budget would allow a single 900^3 volume with 10 bytes per voxel (16-bit SDF, 16-bit SDF-weight, 2×8-bit foreground/background, 4×8-bit RGB+weight) or, as here, a 256^3 background volume and up to 2.5k object volumes with dimension 64^3, which requires 2MB. Our object volumes dynamically vary up to 128^3 and on our office sequence used ∼4MB/object, as shown in Figure 6.12. Of course, more efficient alternatives such as an octree or voxel hashing can also be used to directly eliminate wasted free-space voxels, and these methods are also directly applicable to our approach.

Runtime performance: Our system, although not real-time, scales well with the number of objects. Excluding relocalisation on the office sequence, the frame rate performance against the number of objects was between 10-13Hz (shown in Figure 6.12). Table 6.2 shows that the performance of many components of the system scale with the number of *visible* objects, and so because the number of objects visible in any given frame is dictated more by the viewpoint than by the total number of objects in the map, these components have relatively constant performance. The driving force behind the linear scaling witnessed in Figure 6.12 is the cost of filtering for the visible set of TSDFs, which costs an additional +0.3ms per object. The TSDF visibility checks were coded in Python and this cost only became particularly evident when larger numbers of objects were included in the map. In future this cost could be reduced by moving to a C++ implementation for visibility checks.

Table 6.2: Run-time analysis of system components (ms) with approximate scaling performance on the indoor office sequence.

Component	**Base (ms)**	**Scaling**
Every frame		
Tracking + coarse TSDF	35	constant
Filter for visible TSDFs	-	+0.3/object
Raycast all TSDFs	25	+0.2/vis. object
Object integration	-	+0.7/vis. object
On detection frames		
Mask R-CNN thread	260	constant
Detection point-cloud	10	constant
New object initialisation	-	+30/new object
Object resize+mask fuse	-	+20/vis. object
TSDF reset/re-localisation		
Relocalisation	780	+65/snapshot
Pose-graph optimisation	80	+2/object

6.6 Conclusion

In this chapter we have explored a method to perform instance mapping and classification of numerous objects of previously unknown shape in real, cluttered indoor scenes. Our online system is built from separate modules designed for image-based instance segmentation, TSDF fusion and tracking, and pose graph optimisation. It makes a persistent map which focuses on object elements of a scene with variable, object-size dependent resolution.

Although a step in the direction of object-oriented mapping, a number of shortcomings became apparent through experimentation with this approach. There is a balance that must be struck between filtering detections and providing good coverage of a scene, and even with the existence probability and deletion mechanism detailed here, over time spurious detections result in a growing clutter of partial object reconstructions. More thorough precision/recall evaluations would be useful for benchmarking further refinements to this aspect of the system. A learned mechanism for filtering and reconstructing these objects, such as [Dai et al., 2018] may prove useful in this regard, or combining view-based segmentation and classification with 3D methods which take advantage of object databases such as ShapeNet [Chang et al., 2015].

There are also important caveats relating to coverage when using the instances within the tracking and localisation system itself. If, for example, the CNN is trained only to detect a single object of interest which is small in size and infrequent in the scene, this will have a detrimental effect on the value of the map for localisation. One way to approach this is to always aim for a large coverage of objects to be viewed in the scene regardless of their worth for the task at hand, and then use only the subset of interest. Another approach in cases with degenerate objects would be to perform tracking and mapping independently of the objects in the same manner as Meaningful Maps [Sünderhauf et al., 2017].

Beyond simply tracking, some important scene components are missing from the map which are necessary for long-range path planning outside of the local background TSDF. Although it is possible to represent these objects with volumetric reconstructions, more efficient and parametrised models such as occupancy floor planes or 'door portals' as part of topographic maps may instead better help to store the essential information for path planning. Work such as that of [Sommer and Cremers, 2018] combines primitive types with SDF map objects and it could be very useful in this regard in future work.

There is also significant scope in future to better combine information from multiple duplicate objects seen from very different views to then reconstruct a single better model, rather than maintaining separate TSDFs for each. The complication with this is that many objects, although similar in appearance, are geometrically adjustable. Chairs can be raised, swivel bases can be rotated, monitors can be angled. If one were to naively reconstruct many adjusted objects into a single TSDF the result would be a significantly worse reconstruction. A more flexible parametrisation with deformable parts and learned degrees of freedom could be useful in future work [Fish et al., 2014], and a number of recent Deep Learned approaches which operate on voxel grids could usefully be applied as well [Wu et al., 2016a].

Finally, in this chapter, as in the rest of this thesis, we have restricted our attention to static indoor scenes. If one excludes people (and the objects they are in contact with) this is a reasonable assumption for many indoor domestic scenes. There has been some recent work on outdoor scenes which specifically mask out semantic areas which break the static scene assumption [Kaneko et al., 2018] and such an approach could also be used in domestic scenes. However longer term, this exclusion approach is unsatisfactory as dynamic objects in a scene may also be particularly important elements of a map for a given task. Our object-oriented representation can naturally be extended to model dynamic rigid objects with individually changing poses and also provides the ability to perform life-long mapping by updating object poses as things are moved between scans.

Chapter 7

Conclusions

Contents

7.1 Contributions

In this thesis we have presented a number of contributions towards 3D indoor scene understanding by combining SLAM techniques with deep learned semantic predictions. We have developed and experimented with two different SLAM systems that aim to incorporate semantic information within quite different map representations. Although very different in nature, a common theme for both of these systems is the focus on real-time capable methods as a practical necessity for many applications of interest ranging from robotics to augmented reality. To this end they have both been developed to make heavy use of GPU computing hardware both for the SLAM systems and for the deep learned semantic components. While not all of the systems presented here achieve real-time performance those that do not are still completely online and have been shown to be near real-time without significant optimisation.

The first semantic map representation we explored was that of SemanticFusion presented in Chapter 4. It mirrored the dense reconstruction approach of 'reconstruct everything' with the semantic equivalent of 'label everything.' The map representation therefore closely followed the surfel formulation of its underlying SLAM system ElasticFusion, which allowed for a simple and efficient update scheme that could operate in real-time. The surfel-based surface representation forms a very natural container in which to also store semantic probability distributions. They form a small surface patch that allows integration of multiple semantic predictions into a single discrete unit, while also providing spatial granularity for semantic segmentation. ElasticFusion's map deformation approach to loop-closure provided a consistent 3D global map and avoided the complications of reintegration of semantic information required in key-frame based approaches.

ElasticFusion is a real-time SLAM system with much of the data and processing occurring on the GPU and our chosen CNN was capable of performing a prediction for a given input image in 50ms also on GPU. To allow the system to operate in real-time we designed the map updates to also occur on the GPU to both minimise device-host transfers and take advantage of the updates highly parallel nature as the surfels are treated independently. This alone was not sufficient to achieve real-time performance given the cost of a CNN prediction and so we also skip a certain number of frames between predictions. The camera motion between a single frame tends to be relatively small and the predictions are quite similar and we found that skipping frames did not cause significant deterioration of the system's performance.

Our experiments in SemanticFusion showed that combining a CNN with a dense SLAM system could lead to improvements in the semantic segmentation accuracy, but the limits of the approach also became clear. The annotated surfels of SemanticFusion operate independently and lack any form of higher-level grouping. To answer simple map queries such as 'How many chairs do we have in the conference room?' would require post-processing to cluster the many surfels with the chair label into coherent groups. Also, because the semantic map lacks instance associations between surfels, inconsistencies can occur such as a single object with many different semantic labels. The process of first reconstructing the entire scene geometry and only then performing instance segmentation means that object geometry cannot be preserved or prioritised in the event of a loop-closure.

To overcome these limitations we explore the new paradigm of object-level mapping by developing the Fusion++ system described in Chapter 6. Fusion++ is a SLAM system designed from the ground up to operate on object instances. This approach has been called object-oriented (or object-level) mapping and we believe it is a natural and efficient way to represent much of the data that is important for robotic scene understanding and interaction as well as human-robot communication. Fusion++ builds a map comprising solely of discovered and reconstructed object instances. Each object is reconstructed within its own individual TSDF volume which allows high-quality reconstructions to be combined with the flexibility of a pose-graph system which does not require the complication of intra-TSDF deformations and which can improve reconstruction quality.

We detect and segment newly discovered object instances by using a deep learned object detection and segmentation architecture called Mask R-CNN. To produce accurate reconstructions, the entirety of the surface geometry is reconstructed within the volume. In order to segment the foreground of the instance a novel 3D analog of a 2D instance mask was used. We found the variability in mask predictions made them unsuitable for multiplicative Bayesian updates, such as that performed in SemanticFusion, and so we used a Beta distribution conjugate prior to robustly fuse 2D instance masks into 3D 'foreground' counts encoded within the TSDF voxels. For each object a single semantic probability distribution is maintained and updated and an 'existence' probability counts missed detections to allow spurious objects to eventually be deleted.

The objects in Fusion++ are used for both tracking and as landmarks in an object-level pose graph. The previous work of SLAM++ used a small pre-reconstructed database of objects to populate a scene and ICP to track them individually. In Fusion++, the variety of object geometries encountered as well as the inherent noise during early stages of reconstruction led ICP to frequently fail to converge. To produce the pose-graph edge measurements we therefore developed a more robust approach which performed ICP on the combined scene geometry and partitioned the resulting residual errors for each object instance.

Fusion++ is designed to operate online and, although not yet a real-time system (operating at 10-13Hz excluding relocalisation on our indoor office scene), we believe that with sufficient software optimisation it can achieve real-time performance on current consumer hardware. A theme of this area of research is providing an easy to use developer interface, and to this end we developed the system itself using a Python front-end with wrappers around the core C++ and CUDA components. The system itself enables large scenes to be tracked with relatively small memory usage and high-fidelity reconstructions by excluding large areas of free-space; up to two thousand object instances can be accommodated on-board a single consumer GPU.

As well as exploring semantics in SLAM systems themselves, we also tackled the problem of data production. Deep learning optimises model parameters on the basis of a gathered dataset. This approach is not unique to deep learning, it is common in all machine learning algorithms, but supervised deep learning has proven to be a particularly effective method to make use of very large datasets with ground truth labels. In the domain of indoor scene understanding with a moving camera the question is, *how do we gather such a dataset?*

While much of the work in this thesis is focused on how SLAM and deep learning can be usefully combined, in Chapter 3 we instead look at how dense SLAM can be used to assist deep learning. Producing a per-pixel semantic segmentation ground truth dataset is notoriously labour intensive. The availability of a dense reconstructed map as well as the camera trajectory within it allows annotations that are made once in 3D to be reprojected into the camera view in order to produce numerous annotated 2D frames of video data. We discuss the development of the 'Object Tagger' software which provides a GUI and a number of tools designed to assist the user in efficiently annotating a 3D surfel map produced by ElasticFusion with ground truth annotations.

The 'Object Tagger' assists in scene annotation, but the process of capturing sequences and annotating them is still sufficiently costly to prohibit large-scale dataset production without substantial resources. In Chapter 5 we explore using photorealistic rendering approaches in order to bypass these costs completely to create an automated synthetic dataset generation system called SceneNet RGB-D. We describe the production of an extremely large (5M images) synthetic dataset of video trajectories which provides pixel-accurate ground truth instance annotations, as well as the camera trajectory, perfect depth, and realistic ray-traced RGB images. We use the dataset to show the performance improvements a synthetic dataset can bring to indoor semantic segmentation, and other researchers have since used it to explore a variety of related problems [Shamwell et al., 2017, Balloch and Chernova, 2017, Chen and Deng, 2018, Bloesch et al., 2018].

We used a physics engine and a large-scale object model dataset to generate physically plausible random scene configurations to improve the variation within the dataset. Previous approaches to trajectory generation, such as inserting manually hand-captured trajectories [Handa et al., 2014] into the scene are unsuitable for such large-scale dataset generation. It lacks variety and is unaware of potential collisions with scene geometry. We found a completely random trajectory generation approach produced trajectories that were a poor approximation of hand-held trajectories in real datasets. To alleviate these issues we propose a novel two-body random trajectory method, consisting of a camera position and a look-at point which operates as a proxy for a point of focus. This approach has since been modified and refined in the InteriorNet dataset [Li et al., 2018], for example by correcting the overly smooth trajectories produced by the original two-body method by augmenting it with a learned generative model to produce more realistic camera shake.

7.2 Discussion and Future Research

The methods explored in this thesis provide a general and useful foundation for combining SLAM with Deep Learning for indoor scene understanding from which more task specific semantic maps can be developed and refined. There is obviously significant scope in future to further develop the systems themselves. For example engineering effort and optimisation of the Fusion++ could allow it to become a truly real-time system and improving the tracking to include photometric information may allow it to produce trajectory errors more competitive with state-of-the-art methods.

The virtues associated with a given map representation often come with their own set of challenges, and to decide what the best representation is depends heavily on the application under consideration. SemanticFusion provides an annotated reconstruction of an entire environment which can be immediately used for path planning and structural awareness, but lacks information regarding coherent instances in the map. We approached this problem by developing Fusion++ which provides the object-level information directly, but in so doing we also gave up the complete scene reconstruction. The background TSDF can be used locally for path planning but it is confined to a small volume and cannot provide long distance path planning and large scene navigation. The two approaches could be combined by saving and stitching together the background TSDFs into a complete reconstruction. Another approach more in keeping with the minimum requirements of navigation could be to augment the system with flexible topographic maps of the navigable space that work flexibly with the pose graph structure.

There are a number of features of modern domestic indoor scenes that we have notably not taken full advantage of in the work presented here, and this also offers an opportunity for future improvements. Indoor scenes often include significant structural elements which can be modelled with simple geometric primitives, most notably bounded planar regions. Including this information in a robust manner could greatly enhance the information encoded in simple object-level maps both reducing the memory footprint, improving the map models, and allowing inference about the support surfaces and orientation of objects.

Related to using simple geometric primitives for scene modelling is the potential to explore alternative parametrisations of the objects themselves. In Fusion++ we reconstruct each object individually, but a map often contains repeated elements of nearly identical instances. The slight variability between instances such as reclined chairs or open drawers in a filing cabinet means that a naive approach of simply combining reconstructions will only work for completely rigid objects. A deep learning approach could be to learn the degrees of freedom in each object instance as well as their geometry to form a deformable canonical model. Another possibility, given the ubiquity of mass manufactured products that populate indoor scenes, is that in future large databases of high quality objects may become available to be recognised and downloaded to the map during operation. However, even in this case objects outside of the database or that are highly deformable may still require on-the-fly reconstruction and 'editing' based on available measurements.

This thesis has focused on static scenes, however in domestic environments there are clearly scene dynamics that are important and must be captured. Most notably humans and the objects they interact with are an important element of a map that must be modelled by robotic agents operating in the same environment, for safety considerations if nothing else. The object-level representation provides a natural way to model rigid dynamic objects in a scene, but deformable human models would be required to capture the motion of people. Humans have an intuitive understanding of the properties of more complex entities in indoor scenes as well such as clothes and liquids, and to attain human-level understanding of a scene requires these properties must also be encoded in the map. Work on this could enable augmented reality applications and life-long maps that can be updated as scenes change over time.

Here we have explored using synthetic data as a means of producing training data for supervised training for indoor semantic segmentation. Further work to produce interactive and dynamic synthetic environments could provide a useful means of training and validating different approaches to indoor scene understanding by domestic agents that is difficult or very costly to achieve in the physical world. In our work we found that to transfer to the real world, datasets still required fine-tuning on real world images. This reality gap is an important limitation of current approaches to synthetic training data which must be better understood and mitigated in order to allow synthetic training to reach its true potential. Recent approaches to domain transfer such as Generative Adversarial Networks (GANs) and Domain Separation Networks provide modern tools to tackle the challenges in this area, but it is by no means a solved problem.

Finally, the modular approach of combining independent SLAM and deep learning components taken here is by no means the only approach. Components of the SLAM systems used here may greatly benefit from the robustness of having learned approaches embedded to a greater degree. The map representations explored here have followed a classical metric approach to mapping. The versatility of deep learning also provides the potential for more drastically alternative approaches to mapping. An example of this is to have the ‘map’ stored within the weights of a Neural Network and queried for the desired information while allowing the internal representation itself to be entirely learned. This operation could be seen as akin to asking a human to sketch a room layout. Should the system correctly answer the sufficiently detailed and varied questions we can assume it has properly understood the scene.

Bibliography

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. `http://download.tensorflow.org/paper/whitepaper2015.pdf`. 58

[Arbeláez et al., 2014] Arbeláez, P., Pont-Tuset, J., Barron, J., Marques, F., and Malik, J. (2014). Multiscale combinatorial grouping. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 144

[Armeni et al., 2017] Armeni, I., Sax, A., Zamir, A. R., and Savarese, S. (2017). Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *arXiv preprint arXiv:1702.01105*. 68, 112

[Aubry et al., 2014] Aubry, M., Maturana, D., Efros, A., Russell, B., and Sivic, J. (2014). Seeing 3D chairs: exemplar part-based 2D-3D alignment using a large dataset of CAD models. In *CVPR*. 110

[Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*. 47

[Bailey and Durrant-Whyte, 2006] Bailey, T. and Durrant-Whyte, H. (2006). Simultaneous Localisation and Mapping (SLAM): Part II. *IEEE Robotics and Automation Magazine*, 13(3):108–117. 9

[Balloch and Chernova, 2017] Balloch, J. C. and Chernova, S. (2017). An RGBD segmentation model for robot vision learned from synthetic data. In *Proceedings of the Workshop on Spatial-Semantic Representations in Robotics at Robotics: Science and Systems (RSS)*. 7, 134, 172

[Bao et al., 2012] Bao, S. Y., Bagra, M., Chao, Y.-W., and Savarese, S. (2012). Semantic Structure From Motion with Points, Regions, and Objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 14

[Barron et al., 1994] Barron, J., Fleet, D., and Beauchemin, S. (1994). Performance of optical flow techniques. *International Journal of Computer Vision (IJCV)*, 12:43–77. 110

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). SURF: Speeded up robust features. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 11

[Bengio et al., 2001] Bengio, Y., Ducharme, R., and Vincent, P. (2001). A neural probabilistic language model. In *Neural Information Processing Systems (NIPS)*. 42

[Bengio et al., 2009] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*. 133

[Bengtsson and Baerveldt, 2003] Bengtsson, O. and Baerveldt, A. J. (2003). Robot localization based on scan-matching—estimating the covariance matrix for the IDC algorithm. *Robotics and Autonomous Systems*, 44:29–49. 160

[Bergen et al., 1992] Bergen, J. R., Anandan, P., Hanna, K. J., and Hingorani, R. (1992). Hierarchical model-based motion estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 36

[Besl and McKay, 1992] Besl, P. and McKay, N. (1992). A method for Registration of 3D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2):239–256. 34

[Blais and Levine, 1995] Blais, G. and Levine, M. D. (1995). Registering Multiview Range Data to Create 3D Computer Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 17(8):820–824. 34

[Blanco and Rai, 2014] Blanco, J. L. and Rai, P. K. (2014). nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) wih kd-trees. `https://github.com/jlblancoc/nanoflann`. 73

[Bloesch et al., 2018] Bloesch, M., Czarnowski, J., Clark, R., Leutenegger, S., and Davison, A. J. (2018). CodeSLAM — learning a compact, optimisable representation for dense visual SLAM. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5, 7, 134, 172

[Bloesch et al., 2016] Bloesch, M., Sommer, H., Laidlow, T., Burri, M., Nützi, G., Fankhauser, P., Bellicoso, D., Gehring, C., Leutenegger, S., Hutter, M., and Siegwart, R. (2016). A Primer on the Differential Calculus of 3D Orientations. *CoRR*, abs/1606.0. 26

[Boykov and Kolomogorov, 2004] Boykov, Y. and Kolomogorov, V. (2004). An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(9):1124–1137. 74

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint:1606.01540*. 6

[Brostow et al., 2009] Brostow, G., Fauqueur, J., and Cipolla, R. (2009). Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88 – 97. 62, 63

[Brostow et al., 2008] Brostow, G., Shotton, J., Fauqueur, J., and Cipolla., R. (2008). Segmentation and Recognition using Structure from Motion Point Clouds. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 14

[Butler et al., 2012] Butler, D. J., Wulff, J., Stanley, G. B., and Black, M. J. (2012). A naturalistic open source movie for optical flow evaluation. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 110

[Cadena et al., 2016] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. J. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics (T-RO)*, 32(6):1309–1332. 9, 10, 13

[Calli et al., 2015] Calli, B., Singh, A., Walsman, A., Srinivasa S. and, Abbeel, P., and Dollar, A. M. (2015). The ycb object and model set: Towards common benchmarks for manipulation research. In *International Conference on Advanced Robotics (ICAR)*, pages 510–517. 163

[Calonder et al., 2010] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). BRIEF: Binary Robust Independent Elementary Features. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 11

[Castellanos, 1998] Castellanos, J. A. (1998). *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*. PhD thesis, Universidad de Zaragoza, Spain. 9

[Castle et al., 2007] Castle, R. O., Gawley, D. J., Klein, G., and Murray, D. W. (2007). Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 15

[Cavallari, 2017] Cavallari, T. (2017). *Semantic SLAM: A New Paradigm for Object Recognition and Scene Reconstruction*. PhD thesis, University of Bologna. 83, 103

[Cavallari and Di Stefano, 2016a] Cavallari, T. and Di Stefano, L. (2016a). Online Large Scale Semantic Fusion. In *ECCV 2016 Workshop on Geometry Meets Deep Learning*. 83

[Cavallari and Di Stefano, 2016b] Cavallari, T. and Di Stefano, L. (2016b). SemanticFusion: Joint Labeling, Tracking and Mapping. In *ECCV 2016 Workshop*. 103

[Cavallari and Di Stefano, 2016c] Cavallari, T. and Di Stefano, L. (2016c). Volume-Based Semantic Labeling with Signed Distance Functions. *Image and Video Technology*, 1:544–556. 15, 83, 139

[Censi, 2007] Censi, A. (2007). An accurate closed-form estimate of ICP's covariance. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 160

[Chang et al., 2017] Chang, A., Dai, A., Funkhouser, T., Halber, M., Niessner, M., Savva, M., Song, S., Zeng, A., and Zhang, Y. (2017). Matterport3D: Learning from RGB-D data in indoor environments. In *Proceedings of the International Conference on 3D Vision (3DV)*. 68

[Chang et al., 2015] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. (2015). ShapeNet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*. 111, 112, 167

[Chatila and Laumond, 1985] Chatila, R. and Laumond, J. (1985). Position referencing and consistent world modeling for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 13

[Chen et al., 2018] Chen, L. C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2018). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 40(4):834–848. 41

[Chen and Deng, 2018] Chen, W. and Deng, J. (2018). Learning Single-Image Depth from Videos using Quality Assessment Networks. *arXiv preprint arXiv:1806.09573*. 7, 134, 172

[Chen and Medioni, 1992] Chen, Y. and Medioni, G. (1992). Object modeling by registration of multiple range images. *Image and Vision Computing (IVC)*, 10(3):145–155. 34

[Cheng et al., 2014] Cheng, M., Zhang, Z., Lin, W., and Torr, P. H. S. (2014). BING: Binarized Normed Gradients for Objectness Estimation at 300fps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 144

[Cheng et al., 2005] Cheng, Y., Maimone, M., and Matthies, L. (2005). Visual odometry on the mars exploration rovers. In *Proceedings of the International Conference on Systems, Man and Cybernetics, (SMC)*, volume 1, pages 903–910. 9

[Choi et al., 2015] Choi, S., Zhou, Q., and Koltun, V. (2015). Robust Reconstruction of Indoor Scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 58, 136

[Choudhary et al., 2014] Choudhary, S., Trevor, A. J. B., Christensen, H. I., and Dellaert, F. (2014). SLAM with object discovery, modeling and mapping. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 16, 141

[Civera et al., 2011] Civera, J., Gàlvez-Lòpez, D., Riazuelo, L., Tard'øs, J. D., and Montiel, J. M. M. (2011). Towards semantic slam using a monocular camera. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 15

[Collet et al., 2013] Collet, A., Xiong, B., Gurau, C., Hebert, M., and Srinivasa, S. S. (2013). Exploiting Domain Knowledge for Object Discovery. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 141

[Collobert et al., 2011] Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A Matlab-like Environment for Machine Learning. In *Neural Information Processing Systems (NIPS)*. 58, 129

[Couprie et al., 2013] Couprie, C., Farabet, C., Najman, L., and LeCun, Y. (2013). Indoor semantic segmentation using depth information. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 84, 94, 128

[Curless and Levoy, 1996] Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of SIGGRAPH*. 3, 12, 136, 140

[Czarnowski et al., 2017] Czarnowski, J., Leutenegger, S., and Davison, A. J. (2017). Semantic texture for robust dense tracking. In *Proceedings of the International Conference on Computer Vision Workshops (ICCVW)*. 103

[Dai et al., 2018] Dai, A., , Sturm, J., and Nießner, M. (2018). Scancomplete: Large-scale scene completion and semantic segmentation for 3d scans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 167

[Dai et al., 2017a] Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., and Nießner, M. (2017a). ScanNet: Richly-annotated 3d reconstructions of indoor scene. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 66, 67, 78, 112

[Dai et al., 2017b] Dai, A., Nießner, M., Zollhöfer, M., Izadi, S., and Theobalt, C. (2017b). BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Re-integration. *ACM Transactions on Graphics (TOG)*, 36(3):24:1–24:18. 58, 67, 136

[Davison, 1998] Davison, A. J. (1998). *Mobile Robot Navigation Using Active Vision*. PhD thesis, University of Oxford. 9

[Davison, 2003] Davison, A. J. (2003). Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 10

[DeSouza et al., 2017] DeSouza, C. R., Gaidon, A., Cabon, Y., and López Peña, A. M. (2017). Procedural generation of videos to train deep action recognition networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 110

[Dharmasiri et al., 2016] Dharmasiri, T., Lui, V., and Drummond, T. (2016). MO-SLAM: Multi Object SLAM with Run-Time Object Discovery through Duplicates. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 142

[Drummond, 2014] Drummond, T. (2014). Lie groups, lie algebras, projective geometry and optimization for 3d geometry, engineering and computer vision. http://twd20g.blogspot.com/p/notes-on-lie-groups.html. 26

[Durrant-Whyte and Bailey, 2006] Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms. *IEEE Robotics and Automation Magazine*, 13(2):99–110. 9

[Durrant-Whyte, 1988] Durrant-Whyte, H. F. (1988). Uncertain Geometry in Robotics. *International Journal of Robotics Research (IJRR)*, 4(1):23–31. 9

[Eade, 2009] Eade, E. (2009). Gauss-Newton / Levenberg-Marquardt Optimization. http://www.ethaneade.com/optimization.pdf. 32

[Eade, 2014] Eade, E. (2014). Lie groups for computer vision. http://www.ethaneade.com/lie_groups.pdf. 26

[Eigen and Fergus, 2015] Eigen, D. and Fergus, R. (2015). Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional

Architecture. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 84, 85, 87, 97, 98, 99, 100, 102

[Endres et al., 2014] Endres, F., Hess, J., Sturm, J., Cremers, D., and Burgard, W. (2014). 3D Mapping with an RGB-D Camera. *IEEE Transactions on Robotics (T-RO)*, 30(1):177–187. 65

[Engel et al., 2017] Engel, J., Koltun, V., and Cremers, D. (2017). Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*. 11

[Erhan et al., 2014] Erhan, D., Szegedy, C., Toshev, A., and Anguelov, D. (2014). Scalable object detection using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 19, 144, 145

[Everingham et al., 2010] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision (IJCV)*, 2:303–338. 54, 63, 84, 94

[Fauqueur et al., 2007] Fauqueur, J., Brostow, G., and Cipolla, R. (2007). Assisted video object labeling by joint tracking of regions and keypoints. In *ICCV Workshops*. 65

[Fehr et al., 2017] Fehr, M., Furrer, F., Ivan, D., Sturm, J., Gilitschenski, I., Siegwart, R., and Cadena, C. (2017). TSDF-based change detection for consistent long-term dense reconstruction and dynamic object discovery. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 141

[Felzenszwalb and Huttenlocher, 2004] Felzenszwalb, P. and Huttenlocher, D. (2004). Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision (IJCV)*, 59(2):167–181. 67

[Felzenszwalb et al., 2010] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 32(9):1627–1645. 14

[Finman et al., 2013] Finman, R., Whelan, T., and Kaess, M. (2013). Toward lifelong object segmentation from change detection in dense RGB-D maps. In *Proceedings of the European Conference on Mobile Robotics (ECMR)*. 141

[Fischer et al., 2015] Fischer, P., Dosovitskiy, A., Ilg, E., Häusser, P., Hazırbaş, C., Golkov, V., van der Smagt, P., Cremers, D., and Brox, T. (2015). FlowNet: Learning Optical Flow with Convolutional Networks. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 20, 110

[Fish et al., 2014] Fish, N., Averkiou, A., van Kaick, O., Sorkine-Hornung, O., Cohen-Or, D., and Mitra, N. J. (2014). Meta-representation of shape families. *Transactions on Graphics (Special issue of SIGGRAPH 2014)*. 167

[Fukushima and Miyake, 1982] Fukushima, K. and Miyake, S. (1982). Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 15(6):455–469. 18

[Gaidon et al., 2016] Gaidon, A., Wang, Q., Cabon, Y., and Vig, E. (2016). Virtual Worlds as Proxy for Multi-Object Tracking Analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 110

[Galindo et al., 2005] Galindo, C., Saffiotti, A., Coradeschi, S., Buschka, P., Fernandez-Madrigal, J. A., and Gonzalez, J. (2005). Multi-hierarchical semantic maps for mobile robotics. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 15

[Garcia and Zalevsky, 2007] Garcia, J. and Zalevsky, Z. (2007). Range mapping using speckle decorrelation. US Patent, US7433024B2. 28

[Genova et al., 2017] Genova, K., Savva, M., Chang, A. X., and Funkhouser, T. (2017). Learning Where to Look: Data-Driven Viewpoint Set Selection for 3D Scenes. *arXiv preprint arXiv:1704.02393*. 7, 134

[Girshick, 2015] Girshick, R. (2015). Fast r-cnn. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 148

[Girshick et al., 2014] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 19, 144, 146

[Glocker et al., 2015] Glocker, B., Shotton, J., Criminisi, A., and Izadi, S. (2015). Real-Time RGB-D Camera Relocalization via Randomized Ferns for Keyframe Encoding. *IEEE Transactions on Visualization and Computer Graphics*, 21(5):571–583. 71

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 51

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org. 17, 39, 61

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. 132

[Guennebaud et al., 2010] Guennebaud, G., Jacob, B., et al. (2010). Eigen v3. http://eigen.tuxfamily.org. 36

[Guo et al., 2017] Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. *The Proceedings of Machine Learning Research*. 103

[Gupta et al., 2013] Gupta, S., Arbelaez, P., and Malik, J. (2013). Perceptual organization and recognition of indoor scenes from RGB-D images. In *CVPR*. 162

[Gupta et al., 2015a] Gupta, S., Arbeláez, P. A., Girshick, R. B., and Malik, J. (2015a). Aligning 3D models to RGB-D images of cluttered scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 84

[Gupta et al., 2015b] Gupta, S., Arbeláez, P. A., Girshick, R. B., and Malika, J. (2015b). Aligning 3D Models to RGB-D Images of Cluttered Scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 84, 110

[Gupta et al., 2014] Gupta, S., Girshick, R., Arbelaez, P., and Malik, J. (2014). Learning Rich Features from RGB-D Images for Object Detection and Segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 84

[Gutmann and Konolige, 1999] Gutmann, J.-S. and Konolige, K. (1999). Incremental Mapping of Large Cyclic Environments. In *International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*. 9

[Handa et al., 2012] Handa, A., Newcombe, R. A., Angeli, A., and Davison, A. J. (2012). Real-Time Camera Tracking: When is High Frame-Rate Best? In *Proceedings of the European Conference on Computer Vision (ECCV)*. 6, 110, 119

[Handa et al., 2016] Handa, A., Pătrăucean, V., Badrinarayanan, V., Stent, S., and Cipolla, R. (2016). SceneNet: Understanding Real World Indoor Scenes With Synthetic Data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 66, 84, 111, 112

[Handa et al., 2014] Handa, A., Whelan, T., McDonald, J. B., and Davison, A. J. (2014). A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 6, 109, 110, 119, 172

[Häne et al., 2013] Häne, C., Zach, C., Cohen, A., Angst, R., and Pollefeys, M. (2013). Joint 3d scene reconstruction and class segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 102

[Hazirbas et al., 2016] Hazirbas, C., Ma, L., Domokos, C., and Cremers, D. (2016). FuseNet: incorporating depth into semantic segmentation via fusion-based CNN architecture. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*. 129

[He et al., 2017] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 19, 139, 142, 144, 149, 156

[He et al., 2014] He, K., Zhang, X., Ren, S., and Sun, J. (2014). Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 147

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 2, 51, 127

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 19, 46, 161

[Henry et al., 2010] Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2010). RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*. 12

[Hermans et al., 2014] Hermans, A., Floros, G., and Leibe, B. (2014). Dense 3D semantic mapping of indoor scenes from RGB-D images. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 15, 82, 83, 85, 90, 92, 93, 99, 100, 103

[Hodgkin and Huxley, 1952] Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117:500–544. 17

[Hoffman et al., 2016] Hoffman, J., Gupta, S., Leong, J., S., G., and Darrell, T. (2016). Cross-Modal Adaptation for RGB-D Detection. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 104

[Hosang et al., 2014] Hosang, J., Benenson, R., and Schiele, B. (2014). How good are detection proposals, really? In *Proceedings of the British Machine Vision Conference (BMVC)*. 145

[Hua et al., 2016] Hua, B.-S., Pham, Q.-H., Nguyen, D. T., Tran, M.-K., Yu, L.-F., and Yeung, S.-K. (2016). Scenenn: A scene meshes dataset with annotations. In *Proceedings of the International Conference on 3D Vision (3DV)*. 67, 112

[Hubel and Wiesel, 1962] Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in cat's visual cortex. *Journal of Physiology (London)*, 160:106–154. 18

[Ilg et al., 2017] Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., and Brox, T. (2017). FlowNet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 110

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML)*. 46, 47, 128

[Irani and Anandan, 1999] Irani, M. and Anandan, P. (1999). All About Direct Methods. In *Proceedings of the International Workshop on Vision Algorithms, in association with ICCV*. 11

[Isola et al., 2017] Isola, P., Zhu, J., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 20

[Jensen and Christensen, 2000] Jensen, H. W. and Christensen, N. J. (2000). A practical guide to global illumination using photon maps. *Siggraph 2000 Course 8*. 123

[Jia et al., 2014] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*. 58, 86

[Kahler et al., 2016] Kahler, O., Prisacariu, V. A., and Murray, D. W. (2016). Real-time large-scale dense 3d reconstruction with loop closure. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 58

[Kalman, 1960] Kalman, R. (1960). A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45. 9

[Kaneko et al., 2018] Kaneko, M., Iwami, K., Ogawa, T., Yamasaki, T., and Aizawa, K. (2018). Mask-SLAM: Robust feature-based monocular SLAM by masking using semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 168

[Keller et al., 2013] Keller, M., Lefloch, D., Lambers, M., Izadi, S., Weyrich, T., and Kolb, A. (2013). Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion. In *Proc. of Joint 3DIM/3DPVT Conference (3DV)*. 12, 69, 70, 88

[Kim et al., 2016] Kim, J., Lee, J. K., and Lee, K. M. (2016). Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 20

[Kim et al., 2012] Kim, Y. M., Mitra, N. J., Yan, D.-M., and Guibas, L. (2012). Acquiring 3D Indoor Environments with Variability and Repetition. In *SIGGRAPH Asia*. 16

[Klein and Murray, 2007] Klein, G. and Murray, D. W. (2007). Parallel Tracking and Mapping for Small AR Workspaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 10

[Kneip and Furgale, 2014] Kneip, L. and Furgale, P. (2014). Opengv: A unified and generalized approach to real-time calibrated geometric vision. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 158

[Koppula et al., 2011] Koppula, H. S., Anand, A., Joachims, T., and Saxena, A. (2011). Semantic Labeling of 3D Point Clouds for Indoor Scenes. In *Neural Information Processing Systems (NIPS)*. 14, 65, 83, 84

[Kostavelis and Gasteratos, 2015] Kostavelis, I. and Gasteratos, I. (2015). Semantic mapping for mobile robotics tasks: A survey. *Robotics and Autonomous Systems*, 66:86 – 103. 13

[Krähenbühl and Koltun, 2011] Krähenbühl, P. and Koltun, V. (2011). Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials. In *Neural Information Processing Systems (NIPS)*. 83, 85, 92, 93, 94, 104

[Krizhevsky, 2014] Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*. 146

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *Neural Information Processing Systems (NIPS)*. 19, 61, 144

[Kuipers and Byun, 1991] Kuipers, B. and Byun, Y.-T. (1991). A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, 8:47–63. 13

[Kümmerle et al., 2011] Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). g^2o: A General Framework for Graph Optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 27, 161

[Ladicky et al., 2010] Ladicky, L., Russell, C., Kohli, P., and Torr, P. (2010). Graph cut based inference with co-occurrence statistics. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 14

[Lai et al., 2012] Lai, K., Bo, L., Ren, X., and Fox, D. (2012). Detection-based object labeling in 3d scenes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 16

[Laidlow et al., 2017] Laidlow, T., Bloesch, M., Li, W., and Leutenegger, S. (2017). Dense RGB-D-Inertial SLAM with map deformations. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 2

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep Learning. *Nature*, 521:436–444. 18

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 18, 39

[Leonard and Whyte, 1991] Leonard, J. J. and Whyte, D. H. (1991). Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 9

[Leutenegger et al., 2011] Leutenegger, S., Chli, M., and Siegwart, R. (2011). BRISK: Binary robust invariance scalable keypoints. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 11, 142

[Levin et al., 2004] Levin, A., Lischinski, D., and Weiss, Y. (2004). Colorization using Optimization. In *Proceedings of SIGGRAPH*. 87

[Levine et al., 2016] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1). 20

[Li et al., 2018] Li, W., Saeedi, S., McCormac, J., Clark, R., Tzoumanikas, D., Ye, Q., Huang, Y., Tang, R., and Leutenegger, S. (2018). Interiornet: Mega-scale multi-sensor photo-realistic indoor scenes dataset. In *Proceedings of the British Machine Vision Conference (BMVC)*. 7, 8, 66, 112, 116, 132, 133, 173

[Limketkai et al., 2005] Limketkai, B., Liao, L., and Fox, D. (2005). Relational object maps for mobile robots. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 14

[Lin et al., 2017] Lin, G., Milan, A., Chunhua, S., and Reid, I. (2017). RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation. In

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 41

[Lin et al., 2015] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. (2015). Microsoft COCO: Common objects in context. In *arXiv preprint:1405.0312*. 63

[Lin et al., 2014] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 740–755. 63, 66, 84

[Long et al., 2015] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 19, 40, 41, 84

[Lovegrove, 2011] Lovegrove, S. J. (2011). *Parametric Dense Visual SLAM*. PhD thesis, Imperial College London. 11

[Lowe, 1999] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 11

[Lu and Milios, 1997] Lu, F. and Milios, E. (1997). Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*, 4(4):333–349. 9

[Lucas and Kanade, 1981] Lucas, B. D. and Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 11

[Ma and Sibley, 2014] Ma, L. and Sibley, G. (2014). Unsupervised Dense Object Discovery, Detection, Tracking and Reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 141

[Marcotegui et al., 1999] Marcotegui, B., Zanoguera, F., Correia, P., Rosa, R., Marques, F., Mech, R., and Wollborn, M. (1999). A Video Object Generation Tool Allowing Friendly User Interaction. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*. 65

[Matthies et al., 2007] Matthies, L., Maimone, M., Johnson, A., Cheng, Y., Willson, R., Villalpando, C., Goldberg, S., and Huertas, A. (2007). Computer Vision on Mars. *International Journal of Computer Vision (IJCV)*. 9

[McCormac et al., 2018] McCormac, J., Clark, R., Bloesch, M., Davison, A. J., and Leutenegger, S. (2018). Fusion++:volumetric object-level slam. In *Proceedings of the International Conference on 3D Vision (3DV)*. 3, 4, 20

[McCormac et al., 2017a] McCormac, J., Handa, A., Davison, A. J., and Leutenegger, S. (2017a). SemanticFusion: Dense 3D semantic mapping with convolutional neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 3, 4, 20

[McCormac et al., 2016] McCormac, J., Handa, A., Leutenegger, S., and Davison, A. J. (2016). SceneNet RGB-D: 5m photorealistic images of synthetic indoor trajectories with ground truth. In *arXiv preprint:1612.05079*. 66

[McCormac et al., 2017b] McCormac, J., Handa, A., Leutenegger, S., and Davison, A. J. (2017b). SceneNet RGB-D: Can 5M synthetic images beat generic ImageNet pre-training on indoor segmentation? In *Proceedings of the International Conference on Computer Vision (ICCV)*. 7, 20, 66, 112

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133. 17

[Meyer and Do, 2015] Meyer, G. P. and Do, M. N. (2015). 3d grabcut: interactive foreground extraction for reconstructed 3d scenes. In *Proceedings of Eurographics*. 73, 74

[Microsoft Corp, 2010] Microsoft Corp (2010). Microsoft Kinect. `https://www.xbox.com/en-US/xbox-one/accessories/kinect`. 12

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Neural Information Processing Systems (NIPS)*. 42

[Mishkin et al., 2016] Mishkin, D., Sergievskiy, N., and Matas, J. (2016). Systematic evaluation of CNN advances on the ImageNet. *arXiv preprint arXiv:1606.02228*. 46

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*. 6

[Montemerlo et al., 2002] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 9

[Moravec, 1977] Moravec, H. P. (1977). Towards Automatic Visual Obstacle Avoidance. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2, page 584. 8

[Moravec, 1980] Moravec, H. P. (1980). Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover. Technical Report CMU-RI-TR-3, Carnegie Mellon University, Robotics Institute. 8

[Moutarlier and Chatila, 1989] Moutarlier, P. and Chatila, R. (1989). Stochastic multisensory data fusion for mobile robot location and environement modelling. In *Proceedings of the International Symposium on Robotics Research (ISRR)*. 9

[Mozos et al., 2007] Mozos, Ò., Triebel, R., Jensfelt, P., Rottmann, A., and Burgard, W. (2007). Supervised semantic labeling of places using information extracted from sensor data. *Robotics and Autonomous Systems*, 55(5). 14

[Mur-Artal and Tardós, 2014] Mur-Artal, R. and Tardós, J. D. (2014). ORB-SLAM: Tracking and Mapping Recognizable Features. In *Workshop on Multi View Geometry in Robotics (MVIGRO) - RSS 2014*. 10, 142

[Mur-Artal and Tardós, 2015] Mur-Artal, R. and Tardós, J. D. (2015). Probabilistic Semi-Dense Mapping from Highly Accurate Feature-Based Monocular SLAM. In *Proceedings of Robotics: Science and Systems (RSS)*. 16, 142

[Mur-Artal and Tardós, 2017] Mur-Artal, R. and Tardós, J. D. (2017). ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics (T-RO)*, 33(5):1255–1262. 164

[Newcombe et al., 2011a] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A.

(2011a). KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 12, 83, 140, 141, 152

[Newcombe et al., 2011b] Newcombe, R. A., Lovegrove, S., and Davison, A. J. (2011b). DTAM: Dense Tracking and Mapping in Real-Time. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 11, 34, 136

[Newman, 1999] Newman, P. (1999). *On the Structure and Solution of the Simultaneous Localization and Map Building Problem*. PhD thesis, University of Sydney. 9

[Nicholson et al., 2018] Nicholson, L., Milford, M., and Sünderhauf, N. (2018). QuadricSLAM: Constrained Dual Quadrics from Object Detections as Landmarks in Object-oriented SLAM. *IEEE Robotics and Automation Letters*. 143

[Nießner et al., 2013] Nießner, M., Zollhöfer, M., Izadi, S., and Stamminger, M. (2013). Real-time 3D Reconstruction at Scale using Voxel Hashing. In *Proceedings of SIGGRAPH*. 13, 83

[Noh et al., 2015] Noh, H., Hong, S., and Han, B. (2015). Learning deconvolution network for semantic segmentation. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 41, 84, 86, 94, 102

[Nüchter et al., 2003] Nüchter, A., Surmann, H., Lingemann, K., and Hertzberg, J. (2003). Semantic scene analysis of scanned 3d indoor environments. In *Proceedings of the International Workshop on Vision, Modelling and Visualization (VMV)*. 14

[NVIDIA, 2018] NVIDIA (2018). *Compute Unified Device Architecture-Programming Guide Version 9.2*. 56

[Odena et al., 2016] Odena, A., Dumoulin, V., and Olah, C. (2016). Deconvolution and checkerboard artifacts. *Distill*. 48

[Oquab et al., 2014] Oquab, M., Bottou, L., Laptev, I., and Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 62

[Pedersen, 2013] Pedersen, S. A. (2013). Progressive photon mapping on gpus. Master's thesis, NTNU. 121

[Peng et al., 2015] Peng, X., Sun, B., Ali, K., and Saenko, K. (2015). Learning deep object detectors from 3d models. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 110

[Pham et al., 2018] Pham, Q., Hua, B., Nguyen, D. T., and Yeung, S. (2018). Real-time progressive 3d semantic segmentation for indoor scenes. *arXiv preprint arXiv:1804.00257*. 100, 105, 142

[Pillai and Leonard, 2015] Pillai, S. and Leonard, J. J. (2015). Monocular SLAM Supported Object Recognition. In *Proceedings of Robotics: Science and Systems (RSS)*. 142

[Princeton University, 2010] Princeton University (2010). About WordNet. https://wordnet.princeton.edu. 113

[Prisacariu et al., 2014] Prisacariu, V. A., Kähler, O., Cheng, M., Ren, C. Y., Valentin, J. P. C., Torr, P. H. S., Reid, I. D., and Murray, D. W. (2014). A framework for the volumetric integration of depth images. *CoRR*, abs/1410.0925. 13

[Qiu and Yuille, 2016] Qiu, W. and Yuille, A. (2016). UnrealCV: Connecting computer vision to unreal engine. *arXiv preprint arXiv:1609.01326*. 110

[Ranganathan and Dellaert, 2007] Ranganathan, A. and Dellaert, F. (2007). Semantic Modeling of Places using Objects. In *Proceedings of Robotics: Science and Systems (RSS)*. 15

[Razavian et al., 2014] Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). CNN Features off-the-shelf: an Astounding Baseline for Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 62

[Redmon et al., 2016] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 149

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems (NIPS)*, pages 91–99. 19, 149

[Ren and Lee, 2018] Ren, Z. and Lee, Y. J. (2018). Cross-Domain Self-supervised Multi-task Feature Learning using Synthetic Imagery. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 133

[Richter et al., 2016] Richter, S., Vineet, V., Roth, S., and Koltun, V. (2016). Playing for data: Ground truth from computer games. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 110

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. 41, 128, 129

[Ros et al., 2016] Ros, G., Sellart, L., Materzynska, J., Vazquez, D., and Lopez, A. (2016). The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 110

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386. 17

[Rosten and Drummond, 2006] Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 11

[Rublee et al., 2011] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). ORB: an efficient alternative to SIFT or SURF. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 2564–2571. IEEE. 11, 142

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., McClelland, J. L., and PDP Research Group, C., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, Cambridge, MA, USA. 18, 51

[Rünz and Agapito, 2018] Rünz, M. and Agapito, L. (2018). Maskfusion: Real-time recognition, tracking and reconstruction of multiple moving objects. *CoRR*. 142

[Rusinkiewicz and Levoy, 2001] Rusinkiewicz, S. and Levoy, M. (2001). Efficient Variants of the ICP Algorithm. In *Proceedings of the IEEE International Workshop on 3D Digital Imaging and Modeling (3DIM)*. 34, 35

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252. 2, 19, 61, 148

[Russell et al., 2008] Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. (2008). LabelMe: A Database and Web-Based Tool for Image Annotation. *International Journal of Computer Vision (IJCV)*, 77(1-3). 63

[Rusu et al., 2008] Rusu, R. B., Marton, Z. C., Blodow, N., Dolha, M., and Beetz, M. (2008). Towards 3d point cloud based object maps for household environments. *Robotics and Autonomous Systems*, 56(11):927 – 941. 14

[Salas-Moreno et al., 2014] Salas-Moreno, R. F., Glocker, B., Kelly, P. H. J., and Davison, A. J. (2014). Dense planar SLAM. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 70, 102

[Salas-Moreno et al., 2013] Salas-Moreno, R. F., Newcombe, R. A., Strasdat, H., Kelly, P. H. J., and Davison, A. J. (2013). SLAM++: Simultaneous Localisation and Mapping at the Level of Objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3, 16, 84, 105, 140, 143, 159

[Savva et al., 2014] Savva, M., Chang, A. X., Bernstein, G., Manning, C. D., and Hanrahan, P. (2014). On being the right scale: Sizing large collections of 3D models. In *Proceedings of SIGGRAPH*. 116

[Sener et al., 2016] Sener, O., Song, H. O., Saxena, A., and Savarese, S. (2016). Learning transferrable representations for unsupervised domain adaptation. In *Neural Information Processing Systems (NIPS)*. 133

[Sermanet et al., 2014] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2014). OverFeat: Integrated Recognition, Localization and

Detection using Convolutional Networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 19

[Shafaei et al., 2016] Shafaei, A., Little, J. J., and Schmidt, M. (2016). Play and Learn: Using Video Games to Train Computer Vision Models. In *Proceedings of the British Machine Vision Conference (BMVC)*. 110

[Shamwell et al., 2017] Shamwell, E. J., Nothwang, W. D., and Perlis, D. (2017). Deepefference: Learning to predict the sensory consequences of action through deep correspondence. In *Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*. 7, 134, 172

[Shi and Tomasi, 1994] Shi, J. and Tomasi, C. (1994). Good Features to Track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 10

[Shotton et al., 2006] Shotton, J., Winn, J., Rother, C., and Criminisi, A. (2006). TextonBoost: Joint Appearance, Shape and Context Modeling for Multi-Class Object Segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 62, 63

[Silberman et al., 2012] Silberman, N., Hoiem, D., Kohli, P., and Fergus, R. (2012). Indoor segmentation and support inference from RGBD images. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 63, 64, 66, 84, 87, 98, 111, 112

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489. 2, 20

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 19, 86

[Smith et al., 1988] Smith, R., Self, M., and Cheeseman, P. (1988). A stochastic map for uncertain spatial relationships. In *Proceedings of the International Symposium on Robotics Research (ISRR)*. 9

[Smith and Cheeseman, 1986] Smith, R. C. and Cheeseman, P. (1986). On the Representation and Estimation of Spatial Uncertainty. *International Journal of Robotics Research (IJRR)*, 5(4):56–68. 9

[Socher et al., 2013] Socher, R., Ganjoo, M., Manning, C. D., and Ng, A. (2013). Zero-shot learning through cross-modal transfer. In *Neural Information Processing Systems (NIPS)*. 42

[Sommer and Cremers, 2018] Sommer, C. and Cremers, D. (2018). Joint representation of primitive and non-primitive objects for 3d vision. In *Proceedings of the International Conference on 3D Vision (3DV)*. 143, 167

[Song et al., 2015] Song, S., Lichtenberg, S. P., and Xiao, J. (2015). SUN RGB-D: A RGB-D scene understanding benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 567–576. 66, 84, 98, 111, 112, 115

[Song et al., 2017] Song, S., Yu, F., Zeng, A., Chang, A. X., Savva, M., and Funkhouser, T. (2017). Semantic Scene Completion from a Single Depth Image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 7, 66, 111, 112

[Strasdat et al., 2012] Strasdat, H., Montiel, J. M. M., and Davison, A. J. (2012). Visual SLAM: Why filter? *Image and Vision Computing (IVC)*, 30(2):65–77. 10

[Stückler and Behnke, 2012] Stückler, J. and Behnke, S. (2012). Model learning and real-time tracking using multi-resolution surfel maps. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 141

[Stückler and Behnke, 2013] Stückler, J. and Behnke, S. (2013). Hierarchical object discovery and dense modelling from motion cues in RGB-D video. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 141

[Stückler et al., 2015] Stückler, J., Waldvogel, B., Schulz, H., and Behnke, S. (2015). Dense Real-Time Mapping of Object-Class Semantics from RGB-D Video. *Journal of Real-Time Image Processing JRTIP*, 10(4):599–609. 15, 82

[Sturm et al., 2012] Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. (2012). A Benchmark for the Evaluation of RGB-D SLAM Systems. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 31, 136, 138

[Sumner et al., 2007] Sumner, R. W., Schmid, J., and Pauly, M. (2007). Embedded deformation for shape manipulation. In *Proceedings of SIGGRAPH*. 71

[Sünderhauf et al., 2017] Sünderhauf, N., Pham, T. T., Latif, Y., Milford, M., and Reid, I. (2017). Meaningful maps with object-oriented semantic mapping. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 16, 142, 167

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 19, 145

[Tateno et al., 2016] Tateno, K., Tombari, F., and Navab, N. (2016). When 2.5D is not enough: Simultaneous reconstruction, segmentation and recognition on dense slam. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 141

[Telea, 2004] Telea, A. (2004). An Image Inpainting Technique Based on the Fast Marching Method. *Journal of Graphics, GPU, & Game Tools*, 9(1):23–34. 87

[Thrun et al., 1999] Thrun, S., Bennewitz, M., Burgard, W., Cremers, A. B., Dellaert, F., Fox, D., Hahnel, D., Rosenberg, C., Roy, N., Schulte, J., and Schulz, D. (1999). MINERVA: a second-generation museum tour-guide robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 13

[Thrun et al., 2005] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. Cambridge: MIT Press. 9

[Tobin et al., 2017] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. *arXiv preprint arXiv:1703.06907*. 133

[Tomasi and Manduchi, 1998] Tomasi, C. and Manduchi, R. (1998). Bilateral Filtering for Gray and Color Images. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 69

[Trevor et al., 2013] Trevor, A., Gedikli, S., Rusu, R., and Christensen, H. (2013). Efficient Organized Point Cloud Segmentation with Connected Components. In *3rd Workshop on Semantic Perception Mapping and Exploration (SPME)*. 141

[Ulyanov et al., 2016a] Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2016a). Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 20

[Ulyanov et al., 2016b] Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2016b). Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*. 47

[Valentin et al., 2013] Valentin, J., Sengupta, S., Warrell, J., Shahrokni, A., and Torr, P. (2013). Mesh Based Semantic Modelling for Indoor and Outdoor Scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 83, 84

[Valentin et al., 2015] Valentin, J., Vineet, V., Cheng, M.-M., Kim, D., Shotton, J., Kohli, P., Nießner, M., Criminisi, A., Izadi, S., and Torr, P. (2015). Semanticpaint: Interactive 3d labeling and learning at your fingertips. *ACM Transactions on Graphics*, 34(5). 67

[van de Sande et al., 2011] van de Sande, K. E., Uijlings, J. R., Gevers, T., and Smeulders, A. W. (2011). Segmentation as selective search for object recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 144

[Whelan et al., 2015a] Whelan, T., Kaess, M., Johannsson, H., Fallon, M. F., Leonard, J. J., and McDonald, J. B. (2015a). Real-time large scale dense RGB-D SLAM with volumetric fusion. *International Journal of Robotics Research (IJRR)*, 34(4-5):598–626. 58, 59, 157

[Whelan et al., 2015b] Whelan, T., Leutenegger, S., Salas-Moreno, R. F., Glocker, B., and Davison, A. J. (2015b). ElasticFusion: Dense SLAM without a pose graph. In *Proceedings of Robotics: Science and Systems (RSS)*. 3, 12, 58, 64, 68, 80, 136, 142, 164

[Whelan et al., 2012] Whelan, T., McDonald, J. B., Kaess, M., Fallon, M., Johannsson, H., and Leonard, J. J. (2012). Kintinuous: Spatially Extended KinectFusion. In *Workshop on RGB-D: Advanced Reasoning with Depth Cameras, in conjunction with Robotics: Science and Systems*. 12, 136, 141

[Wong et al., 2015] Wong, Y.-S., Chu, H.-K., and Mitra, N. J. (2015). SmartAnnotator: An Interactive Tool for Annotating Indoor RGBD Images. *Computer Graphics Forum (Special issue of Eurographics 2015)*. 67

[Wu et al., 2016a] Wu, J., Zhang, C., Xue, T., Freeman, W., and Tenenbaum, J. (2016a). Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Neural Information Processing Systems (NIPS)*. 167

[Wu et al., 2016b] Wu, Y. et al. (2016b). Tensorpack. `https://github.com/tensorpack/`. 139, 161

[Wu and He, 2018] Wu, Y. and He, K. (2018). Group normalization. *arXiv preprint arXiv:1803.08494*. 47

[Xiang and Fox, 2017] Xiang, Y. and Fox, D. (2017). DA-RNN: Semantic mapping with data associated recurrent neural networks. In *Proceedings of Robotics: Science and Systems (RSS)*. 134

[Xiang et al., 2017] Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. (2017). Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*. 163, 164

[Xiao et al., 2010] Xiao, J., Hays, J., Ehinger, K. A., Oliva, A., and Torralba, A. (2010). SUN Database: Large-scale Scene Recognition from Abbey to Zoo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 66

[Xiao et al., 2013] Xiao, J., Owens, A., and Torralba, A. (2013). Sun3d: A database of big spaces reconstructed using sfm and object labels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 66, 77

[Yi et al., 2016] Yi, K., Trulls, E., Lepetit, V., and Fua, P. (2016). LIFT: Learned invariant feature transform. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 11

[Yosinski et al., 2014] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Neural Information Processing Systems (NIPS)*. 62

[Yosinski et al., 2015] Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H. (2015). Understanding neural networks through deep visualization. In *Deep Learning Workshop, Proceedings of the International Conference on Machine Learning (ICML)*. 38

[Zamir et al., 2018] Zamir, A. R., Sax, A., Shen, W. B., Guibas, L. J., Malik, J., and Savarese, S. (2018). Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 62, 68

[Zhang et al., 2017] Zhang, Y., Song, S., Yumer, E., Savva, M., Lee, J.-Y., Jin, H., and Funkhouser, T. (2017). Physically-based rendering for indoor scene understanding using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 7, 66, 111, 112

[Zhou et al., 2018] Zhou, H., Ummenhofer, B., and Brox, T. (2018). Deeptam: Deep tracking and mapping. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 5

[Zhou and Koltun, 2013] Zhou, Q. and Koltun, V. (2013). Dense scene reconstruction with points of interest. In *Proceedings of SIGGRAPH*. 140

[Zhou et al., 2013] Zhou, Q., Miller, S., and Koltun, V. (2013). Elastic Fragments for Dense Scene Reconstruction. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 12, 136

[Zhu et al., 2017] Zhu, J., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 132