Monocular Visual SLAM with Object and Layout Understanding

Shichao Yang

December 2018

Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Sebastian Scherer, CMU RI (Chair)
Michael Kaess, CMU RI
David Wettergreen, CMU RI
Derek Hoiem, UIUC

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

Abstract

Simultaneous localization and mapping (*SLAM*) has been widely used in autonomous robots and virtual reality (VR). It estimates the motion of the sensor and structure of the environment at the same time. Most visual SLAM methods build a sparse or semi-dense map using point features to enable real-time tracking. However, the sparse map is limited for many advanced tasks such as robot navigation and VR interactions, which require a dense and semantic representation of the map such as objects and planes. Therefore, it brings in the ***3D object and layout understanding*** problem.

Most existing work solves these two tasks separately by first building the SLAM map then detecting objects and planes, which may not work if the SLAM map is too sparse or low quality, especially in low-texture scenarios. In this thesis, we propose a novel approach to jointly solve the two tasks in one system using object and plane level SLAM. The resulting map is semantic meaningful, compact with discrete objects, and dense with planes. More importantly, we demonstrate that SLAM pose estimation and object layout understanding can benefit each other. Objects and planes can provide additional constraints to improve SLAM camera pose estimation. Multi-view optimization also greatly improves object detection accuracy compared to the single view.

The approach is composed of two parts: first, for single view, we propose an efficient cuboid object detection and a sparse graphical model for joint object and layout reasoning. It doesn't require the object shape priors and Manhattan room assumptions. Then for the multi-view, the detected objects and planes provide depth initialization for the map, and are also treated as SLAM landmarks optimized together with camera pose by bundle adjustment. Compared to points, these high-level landmarks provide different geometry, scale and semantic constraints to improve SLAM performance, even in dynamic environments.

The algorithm is the first monocular object and plane level SLAM demonstrated to work in large scale diverse environments from indoor to outdoor, static to dynamic, due to the relaxed prior assumption about the environment. It achieves the state-of-the-art camera pose estimation accuracy on monocular KITTI benchmark and some TUM sequences, and also improves the 3D object detection accuracy on these datasets.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Problems

Imagine a robot needs to perform some tasks in an unknown environment, for example, bring some coffee to a person shown in Fig 1.1. To accomplish this task, the robot first needs to know its location and the destination, which is called *state estimation* or ***localization*** problem. After computing a proper path, the robot needs to avoid collision with tables or chairs during the move. To do that, a 3D map of the environment needs to be built from the onboard sensors, which is called ***mapping*** problem. These two sub-problems are coupled with each other because a good 3D map benefits the localization and meanwhile a good state estimation improves the mapping quality. ***Simultaneous Localization and Mapping (SLAM)*** aims to solve them together. In addition to autonomous robots, SLAM is also widely used in augment and virtual reality (AR/VR). Two state-of-the-art monocular SLAM illustrations are shown in Fig 1.2.

Different sensors could be used for SLAM such as monocular RGB camera, stereo camera or laser scanner. Compared to other sensors, monocular camera requires the minimal calibration and can work in any indoor and outdoor environments. Due to its light-weight, low cost and low power consumption, it is quite suitable for the weight constrained devices such as micro aerial vehicles or VR/AR headsets. Therefore in this thesis, we focus on using the most general monocular camera.

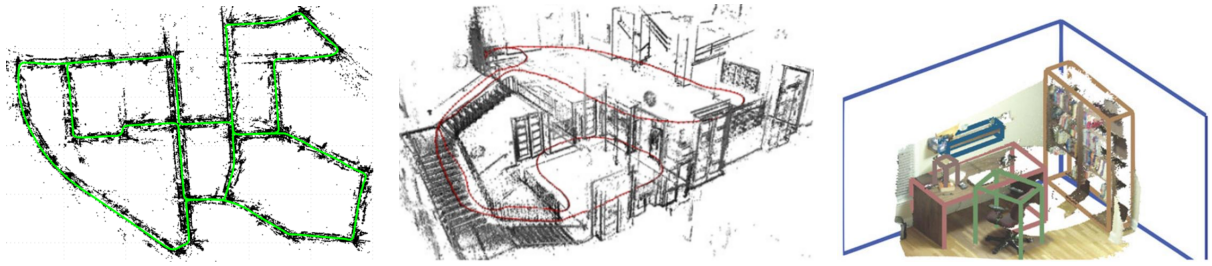
Most visual SLAM algorithms only build a sparse or semi-dense point cloud map which is not enough for many tasks. High level semantic understanding such as the object's category and location is also required. It can greatly improve the robot's intelligence, for examples applications in Fig 1.1:

- For service robots, in addition to pose estimation, they also need to detect and localize 3D interesting objects such as food or table, and room layouts such as wall or ground to interact with humans.
- For AR/VR applications such as placing virtual furnitures in the world, it is necessary to detect the 3D object and plane position to simulate interactions between.
- For autonomous driving, the vehicle needs to detect and localize road surfaces, pedestrians as well as other vehicles in order to take different strategies to avoid them.

This brings to another problem: ***3D object and layout understanding***, abbreviated as 3D

(a) (b) (c)

Figure 1.1: (a) PR2 service robots. Robots need to localize and understand 3D environments. (b) Augmented reality for placing virtual furnitures. The room layout and existing objects need to be detected. (c) Autonomous driving requires the detection of 3D ground plane, cars, and people.



(a) (b) (c)

Figure 1.2: (a,b) Examples of two state-of-the-art geometric visual SLAM: ORB SLAM [1], DSO[2]. (c) Example of 3D object and layout understanding on SUN RGBD dataset [3]

understanding, or scene understanding in this thesis. It is a popular topic in computer vision community especially in recent years due to the popularity of deep learning.

Therefore, in this thesis, we combine SLAM optimization with object and layout understanding and make the following statement:

> A tightly coupled monocular object and plane SLAM can improve both 3D object detection and camera pose estimation.

"Tightly coupled" indicates that object and planes are treated as SLAM landmarks and optimized together with camera poses. It is different from many methods which separately solve SLAM and 3D understanding. This statement also indicates that the two tasks can benefit each other.

In more details, from monocular images, we want to build a 3D semantic map with different meaningful elements such as objects and planes shown in Fig 1.3(b). This map representation has many advantages. First, robots are able to perform more intelligent tasks to interact with the human world. Second, the map is a dense but compact representation of the environment compared to traditional point cloud map thus has better visualization performance. Thirdly, the map is a more complete 3D reconstruction. For instance, most SLAM methods assume the

environment to be static and treat the moving parts as outliers. With the object representation, we can easily track and reconstruct them over time.

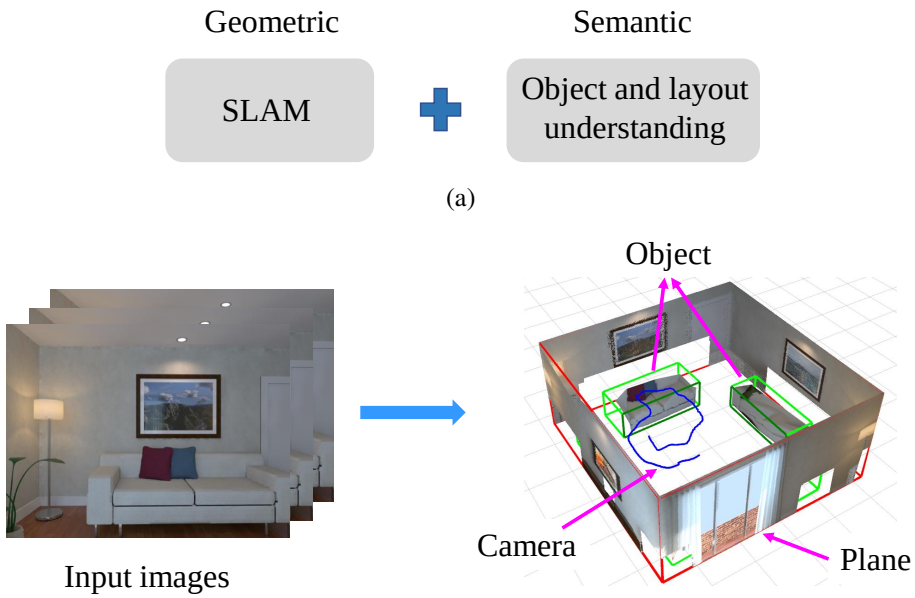


Figure 1.3: (a) Two research topics we are focusing on. Our approach combines SLAM with 3D object and layout understanding. (b) Input and output to our system. The input is a series of images and output is a semantic 3D map composed of layout planes and object cuboids, as well as camera pose estimation.

1.2 Approach and Contribution

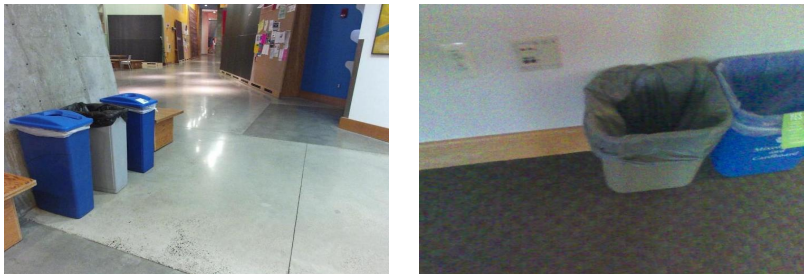
1.2.1 Lack with the state-of-the-art

Though there has been extensive work on SLAM and object layout understanding, there are still many challenges which are briefly summarized here.

For SLAM, state-of-the-art visual SLAM algorithms can achieve impressive results in feature and texture rich environments shown in Fig 1.2. However, for some challenging low-texture environments or highly dynamic scenarios such as Fig 1.4(a), existing point-based SLAM approaches are likely to fail due to the lack of reliable static features. It is also difficult to build a dense and scale consistent map from monocular images.

For object and layout understanding, existing approaches usually rely on the RGBD camera to facilitate the reasoning. The 3D object detection typically depends on object shape priors while the layout understanding mostly assumes a four-wall Manhattan room model with upright viewpoints. These assumptions are likely to break in different structures and abnormal camera viewpoints shown in Fig 1.4(b), which is quite common for robot navigation.

(a)



(b)

Figure 1.4: (a) Challenging scenarios for SLAM. Low texture and dynamic environments. (b) Challenges for object and layout understanding due to diverse scene structures and abnormal camera viewpoints.

1.2.2 Proposed Approach

In this thesis, we propose to jointly solve SLAM and object layout understanding as shown in Fig 1.5. For the *single image*, we first propose a novel cuboid proposal generation method to efficiently detect 3D objects. It doesn't depend on the object shape priors thus is suitable for general scenarios. In addition to object detection, layout planes are also optimized to minimize occlusion and intersection using a high order graphical model. Then for the *multi-view*, the detected 3D objects and planes are treated as SLAM landmarks and optimized with camera poses and point features in unified bundle adjustment (BA) framework. We also propose some novel formulation and observation functions between camera, cuboids, and planes in both static and dynamic environments.

Our main argument is that SLAM and object layout understanding can benefit each other as follows:

(1) One one hand, SLAM improves the accuracy and robustness of 3D understanding. It is difficult to precisely detect the 3D objects just from one image due to occlusions. However, we can refine their positions using multi-view information and SLAM point cloud map. The more accurate camera pose can also improve the single view detections.

(2) On the other hand, object and layout understanding also benefits SLAM pose estimation and mapping. The high-level landmarks such as objects and planes can provide additional semantic, geometric, and long-term scale constraints to improve camera pose estimation. A dense map can also be built with the plane landmarks.

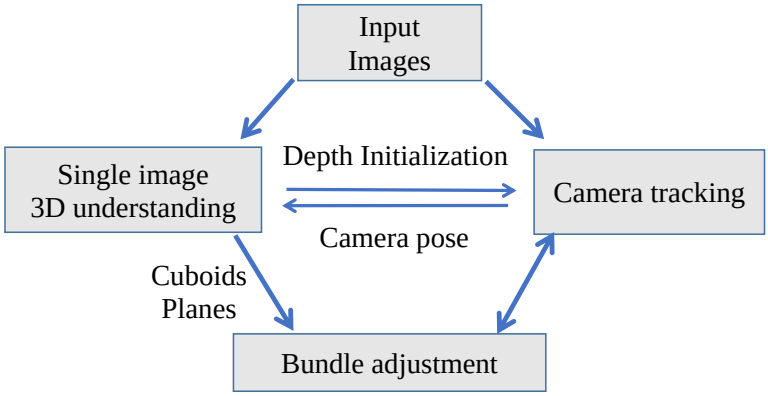


Figure 1.5: The proposed method combing SLAM with object and layout understanding. The detected cuboids and planes from single image understanding are used as SLAM landmarks to improve camera pose estimation. The more accurate camera pose and 3D map in turn, improve the single view detections.

We compare the proposed approach with other representative work in Table 1.1. "Tightly optimization" indicates whether objects and planes' locations are jointly optimized with camera poses. "General scenario" indicates whether it can be applied to general environments instead of only the fixed offline object database or four-wall Manhattan rooms. We can see that **our method is the first object and plane level visual SLAM applicable for general environments**.

In addition to the tight semantic SLAM, we also propose a filtering-based semantic grid mapping system for general scenarios because not all environments can be represented by some discrete objects and layout planes. Apart from using points, objects and planes as SLAM landmarks, we also exploit another geometric feature: edge to improve the state estimation.

Table 1.1: Comparison of thesis related work

Method	Tightly SLAM optimization	Image Only	With Object	With Layout	With/o Shape priors
SsfM [4]	✓	✓	✓		
SLAM++ [5]	✓		✓		
Tsai [6]		✓		✓	
Pillai [7]		✓	✓		✓
Gálvez [8]	✓	✓	✓		
Concha [9]		✓		✓	
Lee [10]	✓			✓	✓
McCormac [11]	✓		✓		✓
QuadriSLAM [12]	✓	✓	✓		✓
Hosseinzadeh [13]	✓		✓	✓	✓
Proposed	✓	✓	✓	✓	✓

1.2.3 Contributions

In summary, the thesis has the following main contributions:

- Propose a new visual SLAM system coupled with 3D object and layout understanding to benefit each other in various static and dynamic environments.
- Develop a bundle adjustment method with novel observation functions to optimize the camera poses, points, cuboid objects and layout planes together.
- Propose an efficient 3D object detection and holistic scene understanding algorithm without shape priors and strict room model assumptions.

1.3 Thesis Outline

There are three main parts of this thesis: single image understanding, tight semantic SLAM and filtering based SLAM shown in Fig 1.6.

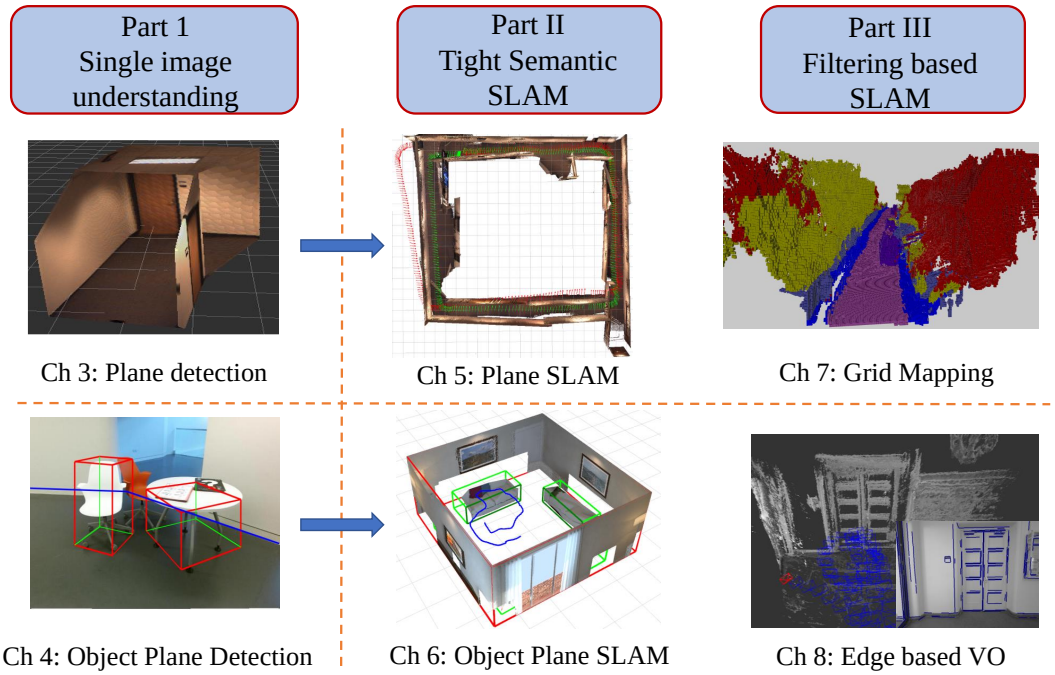


Figure 1.6: Thesis outline. Part I focuses on the single image understanding and Part II extends it to tight Semantic SLAM using objects and planes landmarks. These two parts are the **main contribution parts**. Part III proposes another category of filtering based SLAM using different representations such as grids and edges.

A detailed description of each chapter is as follows:

- Chapter 3 proposes a single image layout plane estimation method using CNNs.
- Chapter 4 presents an algorithm for 3D cuboid object detection and joint object and plane understanding.

- Chapter 5 proposes a pop-up plane SLAM. It extends the single image plane detection to multi-view optimization.
- Chapter 6 presents an object and plane level SLAM. It extends the single image understanding to multi-view optimization in both static and dynamic environments.
- Chapter 7 proposes an generic semantic occupancy grid mapping. It is a real time incremental mapping algorithm using 3D graphical optimization.
- Chapter 8 presents an edge based direct visual odometry. Edges are used to combine both photometric and geometric error in tracking and mapping.

1.4 Publications

This thesis is based on the following peer-reviewed publications:

- **Real-time 3D scene layout from a single image using convolutional neural networks**. Shichao Yang, Daniel Maturana, and Sebastian Scherer. In IEEE international conference on Robotics and automation (ICRA), 2016 `https://youtu.be/2CvFHy5jk1c`
- **Pop-up SLAM: a semantic monocular plane SLAM for low-texture environments**. Shichao Yang, Yu Song, Michael Kaess, and Sebastian Scherer. In IEEE International conference on Intelligent Robots and Systems (IROS), 2016 `https://youtu.be/TOSOWdxmtkw`
- **Robust autonomous flight in constrained and visually degraded shipboard environments**. Zheng Fang, Shichao Yang, *et al.*, and Sebastian Scherer. Journal of Field Robotics (JFR), 2017 `https://youtu.be/XBeiyedW7zE`
- **Direct monocular odometry using points and lines**. Shichao Yang and Sebastian Scherer. In IEEE international conference on Robotics and automation (ICRA), 2017 `https://youtu.be/wu4jL2jQEac`
- **Semantic 3d occupancy mapping through efficient high order CRFs**. Shichao Yang, Yulan Huang, and Sebastian Scherer. In IEEE International Conference on Intelligent Robots and Systems (IROS), 2017 `https://youtu.be/6ylsjkX2YAs`

The following and most relevant publications are under review:

- **CubeSLAM: monocular 3d object SLAM**. Shichao Yang and Sebastian Scherer. arXiv preprint:1806.00557, 2018 `https://youtu.be/QnVlexXi9_c`
- **Monocular object and plane SLAM in structured environments**. Shichao Yang and Sebastian Scherer. arXiv preprint:1809.03415, 2018 `https://youtu.be/jzBMsKCm0uk`

Chapter 2

Background

In this chapter, we explain some background mathematical concepts that are frequently used in the SLAM research, such as nonlinear optimization, observation functions and so on. In addition, we provide a literature review of the related SLAM and object layout understanding work.

2.1 Optimization Problem for SLAM

Notations We utilize the numerator layout convention for matrix and vector calculation. A m dimensional vector is represented as a bold column vector $\mathbf{y} = [y_1, y_2, ..., y_m]^\top$. Each element can be a function of n dimensional input variables $\mathbf{x} = [x_1, x_2, ..., x_n]^\top$. The derivative of a vector $\mathbf{y}$ by vector $\mathbf{x}$ is called the *Jacobian*, which is frequently used in optimization. It is defined as:

$$J_{\mathbf{y}}(\mathbf{x}) = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \nabla y_1(\mathbf{x})^\top \\ \nabla y_2(\mathbf{x})^\top \\ \vdots \\ \nabla y_m(\mathbf{x})^\top \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}_{m \times n} \tag{2.1}$$

2.1.1 Nonlinear Least-Squares Optimization

SLAM is inherently an optimization problem. Thus in this section, we explain the basic nonlinear least squares problem. Given n dimensional variable $\mathbf{x} = [x_1, x_2, ..., x_n]^\top$ and cost function vector composed of m functions $\mathbf{e} = [e_1(\mathbf{x}), e_2(\mathbf{x}), ..., e_m(\mathbf{x})]^\top$. We want to solve the following optimization problem:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{i=1}^{m} \frac{1}{2} e_i^2(\mathbf{x}) = \arg\min_{\mathbf{x}} \frac{1}{2} \mathbf{e}(\mathbf{x})^\top \mathbf{e}(\mathbf{x}) \tag{2.2}$$

$$F(\mathbf{x}) := \frac{1}{2} \mathbf{e}(\mathbf{x})^\top \mathbf{e}(\mathbf{x}) \tag{2.3}$$

$1/2$ is just a coefficient to make following derivation more clear.

There are a variety of methods to solve this problem. The simplest approach is *gradient descent* which always takes steps along the current gradient directions, expressed as: $\mathbf{x}^{k+1} = \mathbf{x}^k - \gamma \nabla F(\mathbf{x}^k)$. It is a first-order iterative algorithm and easy to implement, but the convergence speed might be slow in some problems. Especially when it is close to minimum, the gradient is quite small. An improvement of it is *Newton's method*, which approximates F by a local quadratic function at each iteration and takes a step towards the minimum of the quadratic function. For example, for a quadratic function, this algorithm can find the minimum in one step while gradient descent takes many iterations to proceed. The update rule is $\mathbf{x}^{k+1} = \mathbf{x}^k - [\mathbf{H}_f(\mathbf{x}^k)]^{-1} \nabla F(\mathbf{x}^k)$. However, it is usually intractable to compute second order derivative (also called Hessian) $\mathbf{H}_f(\mathbf{x})$ for general high-dimensional cost functions.

If the cost function F has the form of least-squares costs shown in Eq 2.3, an efficient approximation can be made to the Hessian matrix, which is called *Gauss-newton* algorithm. The SLAM optimization is indeed this kind of problem, therefore, we will explain *Gauss-newton* in more details here.

Gauss-Newton algorithm We first compute the Jacobian matrix of $\mathbf{e}(\mathbf{x})$ denoted as:

$$\mathbf{J}(\mathbf{x}) = \frac{\partial \mathbf{e}}{\partial \mathbf{x}} = \begin{bmatrix} \nabla e_1(\mathbf{x})^\top \\ \nabla e_2(\mathbf{x})^\top \\ \vdots \\ \nabla e_m(\mathbf{x})^\top \end{bmatrix}_{m \times n} \tag{2.4}$$

where $\nabla e_i(\mathbf{x})$ is the gradient of the i-th cost. This also follows the notation in Eq 2.1.

We can then compute the gradient and Hessian of the actual cost function $F(\mathbf{x})$ as:

$$\nabla F(\mathbf{x}) = \sum_{i=1}^{m} e_i(\mathbf{x}) \nabla e_i(\mathbf{x}) = \mathbf{J}(\mathbf{x})^\top \mathbf{e}(\mathbf{x}) \tag{2.5}$$

$$\begin{aligned} \mathbf{H} = \nabla^2 F(\mathbf{x}) &= \sum_{i=1}^{m} \nabla e_i(\mathbf{x}) \nabla e_i(\mathbf{x})^\top + \sum_{i=1}^{m} e_i(\mathbf{x}) \nabla^2 e_i(\mathbf{x}) \\ &= \mathbf{J}(\mathbf{x})^\top \mathbf{J}(\mathbf{x}) + \sum_{i=1}^{m} e_i(\mathbf{x}) \nabla^2 e_i(\mathbf{x}) \\ &\approx \mathbf{J}(\mathbf{x})^\top \mathbf{J}(\mathbf{x}) \end{aligned} \tag{2.6}$$

where in the last step, the second order derivative $\nabla^2 e_i(\mathbf{x})$ is usually much smaller and more difficult to compute compared to first order derivative $\nabla e_i(\mathbf{x})$, therefore, we can simply omit it and approximate $\mathbf{H} \approx \mathbf{J}(\mathbf{x})^\top \mathbf{J}(\mathbf{x})$. This is also the difference between *Newton's method* and *Gauss-newton*.

With the gradient and Hessian, the cost function F around current value $\mathbf{x}$ can be approximated by a quadratic function through Taylor series expansion:

$$F(\mathbf{x} + \Delta\mathbf{x}) \approx F(\mathbf{x}) + \nabla F \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^\top \mathbf{H} \Delta\mathbf{x} \tag{2.7}$$

The optimal update step $\Delta\mathbf{x}$ can be computed by finding the minimum of the above function. This is achieved by setting the first derivative of $F(\mathbf{x}+\Delta\mathbf{x})$ to be zero:

$$\nabla F(\mathbf{x}+\Delta\mathbf{x}) = \nabla F + \mathbf{H}\Delta\mathbf{x} = 0 \tag{2.8}$$

After replacing Eq 2.5 and 2.6 into it, we can get the so-called *normal equation*:

$$\begin{aligned} \mathbf{J}^\top\mathbf{e} + \mathbf{J}^\top\mathbf{J}\Delta\mathbf{x} &= 0 \\ \mathbf{J}^\top\mathbf{J}\Delta\mathbf{x} &= -\mathbf{J}^\top\mathbf{e} \end{aligned} \tag{2.9}$$

We can solve this linear equation and apply the update rule:

$$\begin{aligned} \Delta\mathbf{x} &= -(\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top\mathbf{e} \\ \mathbf{x} &\leftarrow \mathbf{x}+\Delta\mathbf{x} \end{aligned} \tag{2.10}$$

The iteration process is stopped when the error norm drops below a threshold or the maximum number of iterations is reached.

Levenberg-Marquardt (L-M) algorithm As explained before, gradient descent usually converges but has poor performance when it is close to minimum. For Gauss-Newton, the cost may not decrease at each iteration because the quadratic approximation may not always be a good approximation. An alternative method to combine the advantages of both is (L-M) algorithm. The *normal equation* in Eq 2.9 is modified to:

$$(\mathbf{J}^\top\mathbf{J} + \lambda\mathbf{I})\Delta\mathbf{x} = -\mathbf{J}^\top\mathbf{e} \tag{2.11}$$

where λ is a damping ratio and $\mathbf{I}$ is identity matrix. When $\lambda \to \infty$, $\Delta\mathbf{x} \approx -1/\lambda\mathbf{J}^\top\mathbf{e}$ indicating that the update step is along the gradient direction. When $\lambda \to 0$, it changes to the standard Gauss-Newton method.

λ needs to be adjusted in each optimization iteration. If the update $\mathbf{x}+\Delta\mathbf{x}$ reduces the cost, the update is accepted indicating that it is approaching the local minimum, thus λ is reduced to strengthen the influence of Gauss-newton. If the update $\mathbf{x}+\Delta\mathbf{x}$ increases the cost, the update is rejected and λ will be increased which results in a smaller step size and a direction more oriented towards the gradient descent direction.

In all, there are three main steps to solve a nonlinear least square problem iteratively:

- Step 1: Linearize at current variable. Compute the gradient of each cost function.
- Step 2: Build the *normal equation*, and solve for the optimal update.
- Step 3: Update the variables.

2.1.2 Lie Algebra of SO^3 and SE^3

As explained before, the last step for iterative optimization is the variable update shown in Eq 2.10. However, $\mathbf{x} \leftarrow \mathbf{x}+\Delta\mathbf{x}$ only applies to Euclidean space and doesn't apply to non-Euclidean

space such as 3D rotation. A rotation can be minimally parameterized by $\mathbf{x} \in \mathbb{R}^3$, which can be represented by Euler angles or rotation vector. Performing a rotation by $\mathbf{x}$ and then by $\Delta\mathbf{x}$ is in general not equivalent to performing a rotation of $\mathbf{x} + \Delta\mathbf{x}$. Thus rotation cannot be modelled as Euclidean vector space. The correct update rule is to change the vector into rotation matrix then concatenate them as $R(\mathbf{x}) \cdot R(\Delta\mathbf{x})$. In this section, we use Lie group to compute the general update rule for non-Euclidean space. For more details about the mathematical definition of group, manifold, tangent space, Lie algebra, please refer to some tutorial materials [14] [15]. We here briefly introduce some important concepts and formula.

For $\mathbf{SO}_3$ We first discuss the $\mathbf{SO}(3)$ rotation space then extend to $\mathbf{SE}(3)$. The group of $\mathbf{SO}(3)$ has an associated Lie Algebra $\mathfrak{so}(3)$, which has three basis generator matrices, each corresponding to infinitesimal rotations along one axis:

$$G_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, G_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, G_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.12}$$

Consider the axis-angle rotation representation $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]$, we have the corresponding skew symmetric $\mathfrak{so}(3)$ element:

$$[\boldsymbol{\omega}]_\times = \sum_{i=1}^{3} \omega_i G_i = \begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix} \in \mathfrak{so}(3) \tag{2.13}$$

Then exponential map is the tool to map the elements from tangent space (the Lie algebra) to the actual Lie group, for example:

$$\begin{aligned} \exp: \quad \mathfrak{so}(3) &\mapsto \mathbf{SO}(3) \\ \boldsymbol{\omega} &\mapsto \mathbf{R}_{3\times 3} \end{aligned} \tag{2.14}$$

It has the closed form using Rodrigues formula:

$$\exp([\boldsymbol{\omega}]_\times) = \mathbf{I}_3 + \frac{\sin(\theta)}{\theta}[\boldsymbol{\omega}]_\times + \frac{1-\cos(\theta)}{\theta^2}[\boldsymbol{\omega}]_\times^2 \tag{2.15}$$

where the angle $\theta = ||\boldsymbol{\omega}||$. Intuitively, the exponential map generates a rotation by θ radians along the axis direction of $\boldsymbol{\omega}$.

The logarithm map is the inverse of exponential map. It maps the rotation matrix $\mathbf{R}$ to tangent vector $\boldsymbol{\omega}$:

$$\begin{aligned} \theta &= \arccos\left(\frac{\text{tr}(\mathbf{R})-1}{2}\right) \\ \ln(\mathbf{R}) &= \frac{\theta}{2\sin(\theta)}(\mathbf{R} - \mathbf{R}^\top) \\ \boldsymbol{\omega}' &= \lfloor \ln(\mathbf{R}) \rfloor \end{aligned} \tag{2.16}$$

where tr() computes the matrix trace and the operator $\lfloor \cdot \rfloor$ returns the unique off-diagonal elements of a matrix.

For $\mathbf{SE}_3$ $\mathbf{SE}(3)$ is an extension to $\mathbf{SO}(3)$. The transformation matrix is expressed as:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \tag{2.17}$$

The Lie algebra $\mathfrak{se}(3)$ is represented as 6 dimensional vector $\mathbf{v}$ consisting of a translation vector $\mathbf{t} \in \mathbb{R}^3$ and a rotation vector $\boldsymbol{\omega} \in \mathbb{R}^3$ which is same as before:

$$\mathbf{v} = \begin{bmatrix} \mathbf{t} \\ \boldsymbol{\omega} \end{bmatrix} \in \mathbb{R}^6 \tag{2.18}$$

There are six base generators of $\mathfrak{se}(3)$ similar to matrices in Eq 2.12. We can define the hat-operator similar to skew matrix:

$$\widehat{\mathbf{v}} = \begin{bmatrix} 0 & -w_3 & w_2 & t_1 \\ w_3 & 0 & -w_1 & t_2 \\ -w_2 & w_1 & 1 & t_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} [\boldsymbol{\omega}]_\times & \mathbf{t} \\ \mathbf{0}^\top & 0 \end{bmatrix} \tag{2.19}$$

Then the exponential map $\exp:\ \mathfrak{se}(3) \mapsto \mathbf{SE}(3)$ has closed form:

$$\exp(\widehat{\mathbf{v}}) = \begin{bmatrix} \exp([\boldsymbol{\omega}]_\times) & \mathbf{V}\mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \tag{2.20}$$

$$\mathbf{V} = \mathbf{I}_3 + \frac{1-\cos(\theta)}{\theta^2}[\boldsymbol{\omega}]_\times + \frac{\theta - \sin(\theta)}{\theta^3}[\boldsymbol{\omega}]_\times^2 \tag{2.21}$$

where θ and $\exp([\boldsymbol{\omega}]_\times)$ is defined in Eq 2.15.

The logarithm map of $\mathbf{SE}3$ is extended from Eq 2.16:

$$\begin{aligned} \boldsymbol{\omega}' &= \lfloor \ln(\mathbf{R}) \rfloor \\ \mathbf{t}' &= \mathbf{V}^{-1}\mathbf{t} \end{aligned} \tag{2.22}$$

2.1.3 Optimization on $\mathbf{SE}_3$

Combing the previous two sections, we are able to solve the $\mathbf{SE}_3$ optimization problem in SLAM. The SLAM optimization variables are usually poses of the cameras, points, objects and so on. Camera pose T lies in the non-Euclidean $\mathbf{SE}_3$ space, therefore, we need to utilize the Lie algebra explained in the previous section. We still denote $\Delta\mathbf{x}$ as the perturbation update around the current variable $\mathbf{x}$. The variable update formula in Eq 2.10 now changes to a general form:

$$\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x} \implies \mathbf{x}' = \mathbf{x} \boxplus \Delta\mathbf{x} \tag{2.23}$$

More specifically, for the $\mathbf{SE}_3$ variable $\mathbf{x}$, $\Delta\mathbf{x}$ is the tangent vector defined in Eq 2.18, and the update rule becomes:

$$\mathbf{x}' = \mathbf{x} \boxplus \Delta\mathbf{x} \implies \mathbf{x}' = \mathbf{x} \cdot \exp(\widehat{\Delta\mathbf{x}}) \tag{2.24}$$

The cost function Jacobian in Eq 2.4 also needs to extend to the general form:

$$\mathbf{J}(\mathbf{x}) = \left.\frac{\partial \mathbf{e}(\mathbf{x} \boxplus \Delta\mathbf{x})}{\partial \Delta\mathbf{x}}\right|_{\Delta\mathbf{x}=0} \tag{2.25}$$

2.2 Visual SLAM Formulation

2.2.1 Camera geometry

We first define the intrinsic matrix $\mathbf{K}$ of a pinhole camera as:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.26}$$

where f_x, f_y are the camera's focal length in image x, y direction. c_x, c_y are the camera's principal point coordinates. All of them are in image pixel unit.

Then given a point $\mathbf{p} = [p_x, p_y, p_z]^\top$ relative to the camera optical center, its projected pixel $[u, v]$ on the image is:

$$\begin{aligned} \pi : \mathbb{R}^3 &\mapsto \mathbb{R}^2 \\ \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &:= \mathbf{K} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \end{aligned} \tag{2.27}$$

Similarly, given a pixel $[u, v]$ with depth measurement d, we can obtain the 3D position of the point using:

$$\begin{aligned} \pi^{-1} :& \mathbb{R}^2 \times \mathbb{R} \mapsto \mathbb{R}^3 \\ \mathbf{p} =& d\mathbf{K}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \end{aligned} \tag{2.28}$$

2.2.2 Bundle adjustment

We here briefly introduce the specific visual SLAM optimization problem, which is also called bundle adjustment. There are typically two kinds of cost functions for visual SLAM.

Geometric (feature-based) cost function

The feature based methods maintain a map of 3D points and optimize the geometric reprojection error. The camera pose is represented by matrix $\mathbf{T}_{cw} = [R_{cw} \quad t_{cw}; 0 \quad 1]$, from world frame to camera frame. Then given a 3D point P_w in the world frame and its corresponding pixel measurement $\mathbf{x}$ in the image, the reprojection error is defined as:

$$\mathbf{e}_{proj} = \mathbf{x} - \pi(R_{cw}P_w + t_{cw}) \tag{2.29}$$

If there are many map points and camera poses, we can sum all the measurement errors between them and get the final cost function:

$$\underset{R^i_{cw},t^i_{cw},P^j_w}{\arg\min} \sum_{i,j} \| \mathbf{x}^j_i - \pi(R^i_{cw}P^j_w + t^i_{cw}) \|^2_{\Sigma^j_i} \tag{2.30}$$

where $\mathbf{x}^j_i$ is measurement pixel of the 3D point P^j_w in camera i. Σ^j_i is the information matrix relating to the uncertainty of this measurement. The above cost is also in the least square form of Eq 2.2, thus the optimization process explained in the previous section can be utilized.

Photometric (direct) cost function

The direct approaches minimize the photometric intensity error. Instead of maintaining 3D map points, it optimizes some pixels' depth. Given a pixel $\mathbf{x}$ with estimated depth d on camera image i, the photometric error when the pixel is observed on another camera image j is:

$$\mathbf{e}_{photo} = I_i(\mathbf{x}) - I_j\left(\pi(R^j_{cw}(R^i_{wc}\pi^{-1}(\mathbf{x},\mathbf{d}) + t^i_{wc}) + t^j_{cw})\right) \tag{2.31}$$

where we first back-project the pixel with depth from image i then transform and project onto image j. The total errors considering all the observations are summed similar to Eq 2.30.

Another research problem with SLAM is how to solve the linear system in Eq 2.10, more specifically the matrix inversion of $\mathbf{J}^\top\mathbf{J}$. If variable dimension n is high for example with many map points and camera poses, naive matrix inversion takes expensive $\mathcal{O}(n^3)$ computation. Instead, we should utilize the special sparse structures to improve the performance. Variable reordering and Schur complement are commonly used techniques to speed up the computation. Incremental solver is also an effective approach. For more details, please refer to some popular algorithms such as iSAM [16] and g2o [17].

2.3 Related Work

In this section, we discuss the related work of three relevant areas shown in Fig 2.1. We first look at different categories of the classic geometric visual SLAM. Secondly, we review the learning based single image understanding such as object detection and layout understanding. Finally, the most relevant semantic SLAM is discussed in more details.

2.3.1 Geometric SLAM

Our proposed high level object and plane SLAM has difference with point based SLAM but still follows the traditional SLAM pipeline such as data association and graph optimization. Therefore, we first review this area. Depending on the formulation, SLAM is usually grouped as filtering or optimization methods. Based on the measurement functions, SLAM can also be classified as feature based or direct approaches.

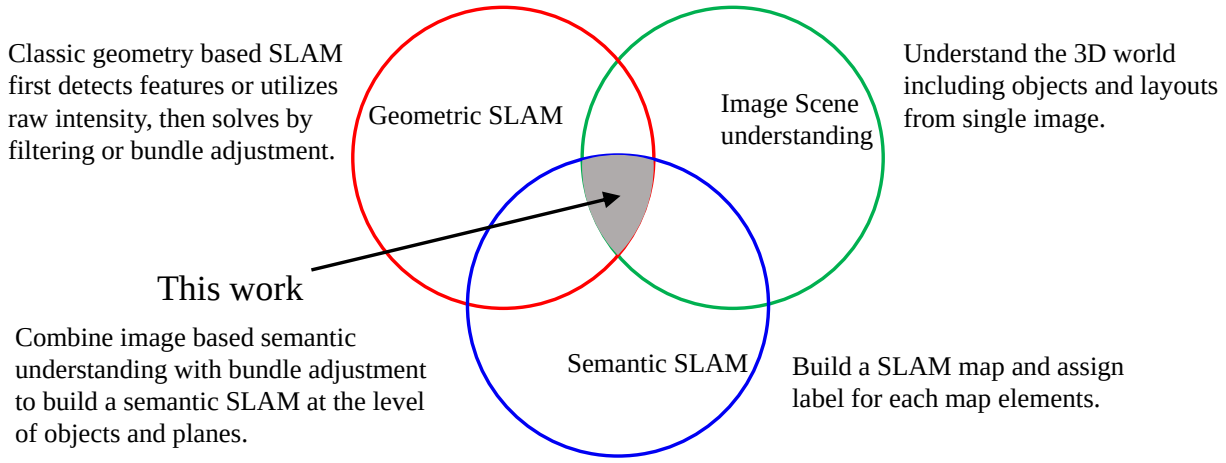


Figure 2.1: Overview of the related work covered in this section and the relationship with our thesis.

Filtering vs. Optimization based SLAM The early stage of SLAM mostly focused on the Bayesian filtering approaches such as Kalman filter or particle filter [18] [19] [20] [21]. It estimates and continuously updates a joint probability distribution of the map and robot poses. It is composed of two steps: one is the state prediction step to propagate the distribution. The second is the measurement update step to correct the distribution. Measurements can come from different sensor data. Theoretically, Kalman filter marginalizes all the old poses and only keeps the current one in the state vector. MonoSLAM [18] is one of the pioneering EKF filtering SLAM methods and is also the first to demonstrate real-time tracking and mapping using a hand-held monocular camera. However, the filtering approaches tend to have large drift in the long term due to the accumulation of linearization errors.

Optimization-based approaches jointly optimize the all the camera poses and map points, which is also called bundle adjustment (BA) [22] or smoothing [23]. However the computation can easily go unbounded as time goes on, therefore in practice, only a subset of camera poses called keyframes are used in the optimization. The optimization is just a nonlinear least square problem and can be solved by Gauss-newton or L-M algorithm using many libraries such as iSAM [16] or g2o [17]. The first representative keyframe-based SLAM is PTAM [24]. It divided the whole SLAM problem into two subtasks of real-time tracking and low-rate mapping, which has become the most popular design of SLAM algorithms such as ORB SLAM [1] and LSD SLAM [25]. A comparison of filtering and optimization-based approaches can be found in [21].

Feature vs. Direct SLAM There are basically two categories of SLAM to utilize the image data. One is called featured based or indirect method. It pre-processes the raw sensor measurements to get intermediate representations such as feature points, lines or planes to be SLAM landmarks. Then the optimization only depends on the extracted features and the geometric cost such as re-projection distance error is usually utilized. This approach greatly simplifies the raw high dimensional image measurement to the low-dimensional geometry features. There are different kinds of features for example FAST corner in PTAM [24], ORB feature in ORB SLAM

[1], SIFT in Fovis[26] and SURF in libviso [27]. In addition to points, lines [28] [29] [30] and planes [31] [32] [33] are also often used in SLAM. In this thesis, we extend it to use high level landmarks such as objects and layout planes.

The second category is called direct method which skips the pre-processing feature extraction step and directly utilizes the raw sensor input such as image pixel intensity. Therefore the optimization cost function is called photometric (intensity) error. Compared to geometric error, photometric error is highly nonlinear and non-convex, therefore it has small convergence basin and requires good initialization of camera poses and map. DTAM [34] is the first real-time dense and direct SLAM method but relies on GPU acceleration. Recently, Engel *et al.* proposed two real-time SLAM running on CPU: LSD-SLAM [25] and DSO[2]. Different from DTAM, LSD and DSO only utilizes some high gradient pixels to speed up.

2.3.2 Object and Layout Detection

In order to build a semantic map, it is necessary to first semantically understand the images. In our thesis, instead of pixel-level semantic segmentation, we are more interested in the high level 3D understanding such 3D object and layout, which is reviewed here.

3D object detection There has been many 2D object detection algorithm using CNNs such as MaskRCNN[35] and Yolo [36]. 3D object detection from a single image [37] [38] is much more challenging because more object pose variables and camera projective geometry need to be considered. Existing approaches can be grouped into two categories depending on whether prior shape models such as PASCAL 3D+ [39] and ShapeNet [40] are available. If the shape prior is represented as an ordered collection of 3D keypoints such as vehicles' headlights or chair's corners, the best object pose can be found through keypoints prediction [41] and Perspective n-Point (PnP) solver [42] [43]. This approach works well for vehicle detection because the keypoints are well defined and all cars have similar shapes. For other shape priors such as CAD model without keypoints annotations, the best pose to align with images is usually found through hand-crafted texture features [44] or more recent deep networks [38] [45].

If there is no shape prior, objects are usually represented by cuboids. The typical approach is to first generate many cuboid proposals then the best one is selected based on various texture features. For example, cuboids can be formed by combinations of Manhattan edges [46][47] or rays through vanishing points [48]. Chen *et al.* proposed to exhaustively sample many 3D boxes on the ground then score them based on context features [49]. Some recent work combines deep learning with projective geometry to improve the detection performance. Two similar work to us [50] [51] finds the best cuboid to fit tightly with the 2D bounding box.

Joint object and plane understanding For layout plane detection, the popular room model based on vanishing points is proposed by Hedau *et al*[52]. Recent learning based approaches such as [53], LayoutNet [54] and RoomNet [55] can achieve impressive results in Manhattan room environments. These approaches can generate roughly correct plane models, but they are not suitable for SLAM landmark optimization because CNN prediction may be inconsistent across frames and cannot be used as measurements. In addition, most of them only apply

to the restricted four-wall room model. Our work is more related to the joint and holistic 3D understanding of object and planes. Their positions can be optimized based on the spatial and semantic relationships such as occlusion, intersection, and concurrence [56][57] [58]. Most of them utilize RGBD camera and are not running in real time. More recent work directly predicts the 3D occupancy of objects and planes by CNN [59].

2.3.3 Semantic SLAM

The most important part of this thesis is a tightly coupled semantic SLAM using high level landmarks. We first review the decoupled approach then discuss the more related coupled semantic SLAM.

Decoupled approach The decoupled approaches first build SLAM point cloud map then further label the points, or detect objects. For the 3D semantic labelling task, the simplest way is to directly transfer the 2D segmentation label to 3D map through pixel back-projection [60]. Some post optimization can be used to improve the smoothness using different graphical models [61] [62] [63]. Deep networks are also designed to directly predict the point cloud's label [64]. Multi-view SLAM and point cloud can be used to improve the object detection performance [65] [7]. Similarly, planes or superpixels are used in [66] [67] [68] to improve dense mapping in low-texture areas. These approaches show improvement compared to 2D labelling or detections, but they might fail if SLAM algorithm cannot build a high quality map.

Coupled SLAM using object and planes The tightly coupled approach utilizes objects and planes as SLAM landmarks to jointly optimize their positions with camera poses. Compared to points, objects and planes can provide long-range geometry and scale constraints. Bao *et al.* proposed the first Semantic SfM to jointly optimize all map elements [4]. [5] proposed a practical real-time SLAM system called SLAM++ using RGB-D cameras and prior object models. Frost *et al.* represented objects as spheres to correct the scale drift of monocular SLAM [69], similarly in [70]. Recently, a real time monocular object SLAM using the prior object models was proposed in [8]. [71] solved multi-view 3D ellipsoid object localization analytically and QuadriSLAM [12] extended it to an online SLAM without prior models. Lee [10] estimated the layout plane and point cloud registration iteratively to reduce RGB-D mapping drift. Hsiao *et al.* used planes to reduce pose drift and create dense mapping for large scale indoor buildings [72]. McCormac *et al.* proposed Fusion++ to build an online volumetric object-level SLAM map without prior shape models using RGBD camera [11]. Recently, [13] proposed a similar work to us which jointly optimized all map components. The difference is that we use monocular camera instead of RGBD camera and also have different object representations and observation functions.

Chapter 3

CNN-based Scene Layout Detection

In this chapter, we propose a method to detect 3D scene layout planes and build a simplified world model from single image using convolutional neural networks. We only consider layout planes in this chapter and more complicated scenarios with objects are addressed in the next chapter.

3.1 Introduction

In order to safely navigate inside corridors, robots need to perceive the environment, detect wall obstacles and generate actions in real time from images. A common approach to this problem is to use geometric 3D reconstruction techniques, but they require feature tracking across multiple images and often fail in low texture environments, which is common in indoor scenes.

Meanwhile, humans can effortlessly extract considerable geometric information from single images. For example, given the image in Figure 3.1 we can quickly interpret the structure and judge the plane distance to the cameras. We wish to provide robots with similar abilities, which should be robust to various conditions such as poor lighting, homogeneous or occlusion. It can greatly extend the sensing horizon beyond stereo cameras and other light-weight active sensors.

Our proposed approach is to combine machine learning with inference of geometric properties to achieve efficient scene understanding. It contains two parts: a learning algorithm to detect ground and wall planes and geometric modelling to build a simplified 3D plane model. For the learning part, we use a type of Convolutional Neural Network (CNN) and a Conditional Random Field (CRF) to predict a geometric layout class for each pixel in the image. We then use geometric constraints to compute the relative orientations of the wall and ground to pop up ground and wall planes into a simplified 3D model. In summary, our main contributions are as follows:

- A real time scene layout prediction combing learning-based semantic segmentation with geometric modelling.
- Outperforms other state-of-the-art methods in terms of accuracy, speed and robustness on various datasets.
- Propose a large corridor dataset with ground annotations.

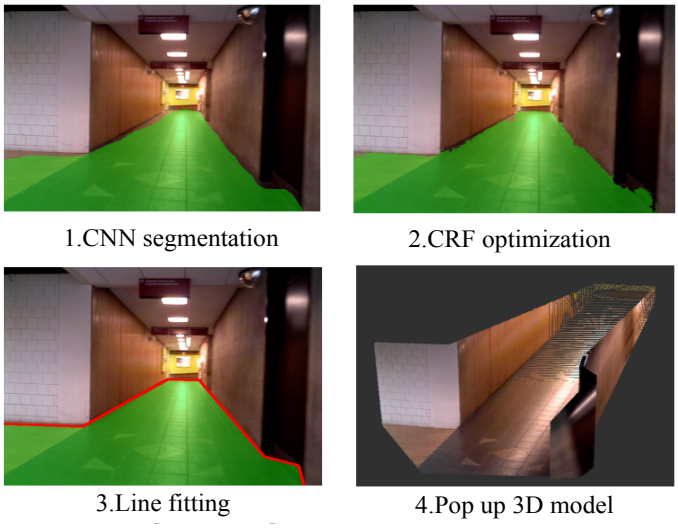


Figure 3.1: Overview of our proposed method. We first segment the image into ground (in green) and walls using CNN, then refine it by CRF. After that, we detect the boundary, fit line segments and pop it up to 3D model.

Closely Related Work

We briefly review the related single and multi-view plane detection at the time of publications. A more complete review can be found in Chapter 2.

Single Image Many works directly predict pixel depth from a single image for example Saxena *et al*'s Make3D [73] system through Markov Random Field optimization, and depth transfer [74] by fetching depth from closest matches in a large database. CNNs can also be used for end-to-end depth prediction such as Eigen *et al*'s Depth-Semantics-Normal (DSN) model [75]. These methods don't consider the scene layout constraints, and thus might yield unreasonable 3D maps.

For the structured room environments, some room layout models are proposed. Lee et al. [76] detected line segments and extend them to generate fixed corridor models. Hedau et al. [52] parameterized room layouts by sampling rays from vanishing points and select the best candidate model based on the surface labels or orientation maps. These methods typically rely on the Manhattan assumption or specific room environments. Moreover, most of these cannot achieve real-time performance, except for the speed-up implementation in [77].

The most similar work to ours is by Hoiem et al. [78]. They use region-based cues such as color, texture and edges, to classify pixels into multiple geometry classes and then fold them into a 3D model. It could be applied to various environments but it is not obvious how to design effective image cues and it also cannot run in real-time. Our CNN-based segmentation obtains significantly more accurate segmentation and the pop-up process is faster and more robust.

Multiple images With multiple images, Structure from Motion and vSLAM are widely used approaches obtain 3D reconstructions. They usually track point features across multiple frames

and build a globally consistent 3D map [24]. These methods are mature but not suitable for many corridor environments because of the sparse visual and geometric features.

More structured plane representation can also be built. Some works generate many candidate 3D model hypotheses and subsequently update their probability by feature tracking across multiple frames. Tsai et at. [6] built corridor models by connecting three edges with different orientations indicating left, center and right wall. Furlan et al. [79] fitted planes from vSLAM point clouds and update the probability.

Robotics Applications Our goal is to build a simplified plane model and enable safe and robust navigation for robots in corridor environments. Some existing works use image cues to navigate such as following vanishing points direction [80] or detecting wall-floor intersection landmarks [81]. These methods usually work in specific environments with specific viewpoints and are not applicable for corners,object obstruction, curved corridors and poor lighting conditions.

3.2 CNN-based Plane Extraction

3.2.1 CNN Model

CNNs have been shown to achieve state-of-the-art performance in various vision tasks such as object recognition [82] and semantic segmentation [83]. The Fully Convolution Network (FCN) architecture of Long et al. [83] and the DSN architecture of Eigen et al. [75] share the central ideas of using "skip connections" to integrate information across different scales and taking advantage of the convolutional nature of the CNNs to perform pixelwise labeling efficiently.

We design and implement a network based on the ideas of FCN and DSN models shown in Figure 3.2. [1]. We modify the networks to generate faster and finer predictions. *Deconvolutional* layers is utilized to generate prediction in different scales. Compared to the standard AlexNet, we decrease the strides of *convolution* and *pooling* to get larger output size. Compared to FCN, we use `conv1` and `conv4` as fusion layers instead of `conv3` and `conv4`, in order to get more diverse features and a larger output size. All the hidden layers use rectified linear units for activations. Dropout is applied to fully-connected layers `conv6` and `conv7`. The input to the network is $320 \times 240 \times 3$ RGB image and the first scale's output is $1/16$ of the input image size. We bilinearly upsample (*deconvolution*) it to $1/8$ scale and fuse it with `conv4` layer to get the second scale's prediction. Again, we upsample and fuse it with `conv1` layer to get the third $1/4$ scale output. It will be finally upsampled to the desired image size.

For training, we minimize the pixel-wise cross-entropy loss:

$$L(C, C^*) = -\frac{1}{n} \sum_i C_i^* \log(C_i) \tag{3.1}$$

where $C_i = e^{z_i} / \sum_c e^{z_{i,c}}$ is the class softmax probability at pixel i given the CNN convolution output z.

[1] No public code is released at the paper submission time

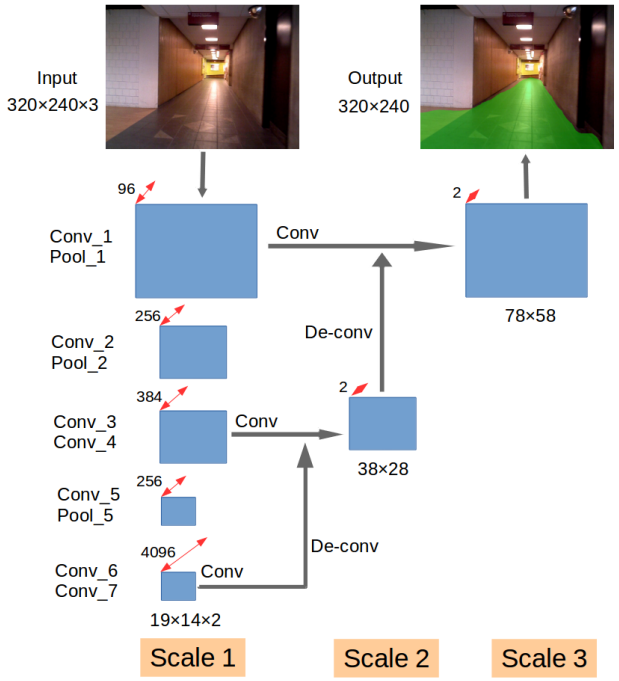


Figure 3.2: Proposed CNN model containing three scales modified based on FCN [83] to generate faster and finer predictions.

3.2.2 CRF model

Our CNN model could effectively predict the geometric layout of the scene. However, it has some shortcomings similar as other CNN models. Due to the relatively coarse output resolution, its predictions do not always capture fine details in the image, as shown in Figure 3.4. Moreover, since we do not explicitly force smoothness, the CNN sometimes creates misclassified patches and discontinuities. This has an adverse effect on the latter line-fitting and pop-up stages of our methods (Section 3.2.3). To fix these problems, we employ a fully connected dense CRF [84] to refine CNN segmentation which allows for long-range constraints among pixels. To make inference tractable, [84] proposes an efficient mean-field inference using Gaussian edge potentials. This technique was also used in conjunction with CNN by [85], showing impressive improvement over pure CNN.

Here we briefly describe this method. Let the prediction label for the n pixels be a vector $\mathbf{x} = (x_1, \cdots, x_n)$. The dense CRF assigns an energy function $E(\mathbf{x})$ to the prediction as a sum of unary and pairwise potentials. The unary potentials are the negative log likelihood of the softmax probabilities from the CNN:

$$\psi_u(x_i) = -\log P(x_i) \tag{3.2}$$

The pairwise potentials enforce consistency between different pixels defined as a weighted sum of Gaussian kernels $\psi_p(x_i, x_j) = \mu(x_i, x_j)\lambda_{i,j}$, where $\mu(x_i, x_j) = 1$ if $x_i \neq x_j$, and $\lambda_{i,j}$ is a function of position p and color intensity I:

$$\lambda_{i,j} = w_1 \exp\left(-\frac{\|p_i - p_j\|^2}{2\sigma_\alpha^2} - \frac{\|I_i - I_j\|^2}{2\sigma_\beta^2}\right) + w_2 \exp\left(-\frac{\|p_i - p_j\|^2}{2\sigma_\gamma^2}\right) \tag{3.3}$$

3.2.3 Pop-Up 3D model

we implemented a faster and simplified image pop-up method compared to Hoiem [78]. We first find ground-wall boundary segments through Douglas-Peucker line simplification [86], which is found to be more robust than Hough Transforms. We then project the ground plane and each wall plane to 3D space by assuming that walls are vertical relative to the ground, i.e. the soft Manhattan assumption [6]. The camera pose and calibration parameters are required for back-projection. Here, we assume the camera is parallel to the ground with a height of h=1 m, which could also be provided from other sensors or extrinsic calibration through vanishing points in Manhattan environments [52].

In more detail, plane is represented as $\pi = (n^T, d)^T \in \mathbb{R}^4$, where n, d is normal vector and distance to origin respectively. Then a 3D point $P = (X, Y, Z)^T$ lies on plane iff $n^T P + d = 0$. P is back-projected from image pixel $p = (x, y, 1)^T$ by $P = \lambda K^{-1} p$, where K is intrinsic calibration matrix. With these two constraints, we can solve for P from p and π:

$$P = \frac{-d}{n^T (K^{-1} p)} K^{-1} p \tag{3.4}$$

Using the assumed pose and ground plane $\pi = (0, 1, 0, 1)$, we first project all ground pixels including the boundary using Equation 3.4. The boundary points also lie on the walls. Using the assumption that wall is vertical to ground, we can thus compute all the plane's equation and project wall plane pixels using Equation 3.4.

3.3 Implementation and Training

3.3.1 Training dataset

This paper focuses on corridor environments, which mobile robots operating indoors often have to traverse. Existing indoor datasets such as the NYU Depth V2 dataset [87] and the SUN RGBD dataset [3] are largely composed by images of cluttered rooms, which are of less interest for our purposes.

To our knowledge, there is no existing large image dataset specifically for corridors. Therefore, we assembled our own dataset[2] for this work. Examples images are shown in Figure 3.3. It contains 967 images from three sources: 349 images from the SUN RGBD [3] (category "corridor"); 327 images from SUN database [88] (category "corridor") and 291 images from self-collected video taken around the Carnegie Mellon University campus. For the SUN database images, we used annotations where available, and manually annotated an extra 250 images using

[2]Dataset is available at `http://theairlab.org/cmu-corridor-dataset/`

Figure 3.3: Corridor dataset created from three sources, containing various scenes with various view points. Left to right: SUN RGBD, SUN Database, self-collected.

LabelMe [89]. For benchmarking purposes, the dataset is split into 725 training images ($\sim 75\%$) and 242 testing images ($\sim 25\%$).

Images are annotated with polygons corresponding to two classes: ground or non-ground (wall). Ceilings were not labelled as they are not important for most robot navigation purposes, but could be easily included if necessary.

3.3.2 CNN Training

We decouple CNN and CRF parameter training, assuming that the unary term in Equation 3.2 computed from the CNN are fixed during CRF parameter searching. This is also the only connection between the CNN and the CRF. There is also some recent works jointly optimizing CNN and CRF [90].

For the CNN training, the parameters are learned through stochastic gradient descent to optimize the cross-entropy loss defined in Equation 3.1. There is no data augmentation such as random flip or rotations. We train the network in stages corresponding to the different scales. The first scale is initialized with the weights of the AlexNet model for the MIT Places205 dataset [91]. Then the first two scales are trained together. Finally, the full three scales are trained together. The batch size is set as 16, learning rate as 0.0001 and momentum as 0.9. Each training process is optimized for 300 epochs until converges. We use the Theano library [92] to compute the gradients and accelerate computation with the GeForce GTX 980 Ti GPU. It takes around five hours to train the network.

3.3.3 CRF parameter searching

CRF hyper parameters $w_1, w_2, \sigma_\alpha, \sigma_\beta, \sigma_\gamma$ in Equation 3.3 are searched by cross-validation on a small subset (100) images to achieve the highest mean IoU (Intersection over Union). Default values of $w_2 = 3$ and $\sigma_\gamma = 3$ are used. The searching ranges of other parameters are: $w_1 \in \{5, 6, \dots 10\}, \sigma_\alpha \in \{2, 6, \dots 12\}, \sigma_\beta \in \{2, 4, \dots 10\}$. The maximum optimization iteration is set as 10 for all the experiments. Since CRF computation complexity grows linearly with pixel

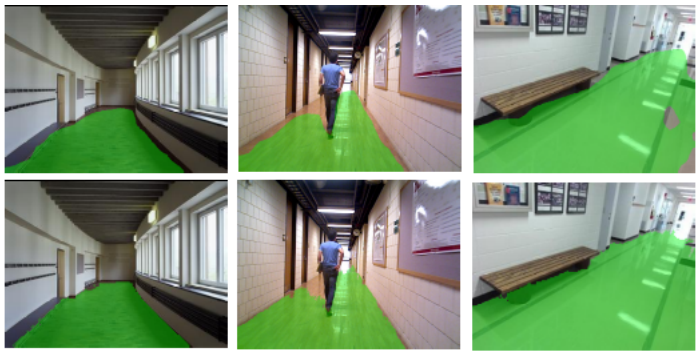


Figure 3.4: CNN prediction and CRF optimization examples on our test dataset. **Row 1**: CNN prediction; **Row 2**: CRF optimization. CNN prediction captures the general location of ground. CRF further improves the spatial consistence and captures fine details.

numbers, we downsample the raw image then upsample the CNN output to 160×120 in order to speed up the prediction. We use the publicly available implementation of dense CRFs [84].

3.4 Experiments

We evaluate the proposed method on both our dataset (242 test images) and public Michigan-Milan Indoor Dataset (84 images) [79]. We adopt three common semantic segmentation metrics: pixelwise accuracy, mean Intersection over Union (IU) and Frequency Weighted IU (F.W. IU) [83]. All predicted labels are upsampled to the raw image size for evaluation.

3.4.1 Evaluation on our mixture data

Qualitative Results

We first qualitatively show the performance of the three main steps in our method. Examples of CNN prediction and CRF optimization are shown in Figure 3.4. We can see that CRF can refine the boundary, remove the extra misclassified ground regions and discontinuous hollow patches. Examples of 3D Pop-Up models are shown in Figure 3.5. Our algorithm works quite well in various corridor types with various lighting conditions and obstructions. To demonstrate the potential for robots' navigation, we apply our algorithm on a video where we pop up a 3D model for each frame independently. More results are provided in the supplementary materials.

Finally, we compare with some other state-of-art methods: Lee *et al.*[76] , Hoiem *et al.* [78] and Hedau *et al.* [52]. in Figure 3.6. Our method works better, especially in curved, homogeneous, and poorly lit corridors.

Quantitative Results

We also report the quantitative results of each step in Table 3.1. The number next to the name is image size for operations in each step. Pop-up accuracy in the last row is the evaluation of the

Figure 3.5: 3D pop up examples from our test dataset. The first row is the raw image and the second row is the pop-up 3D model. Since our algorithm doesn't predict the ceiling plane, we manually remove points above a constant height threshold just for visualization. Since we use assumed a camera pose, the 3D model is up to scale.

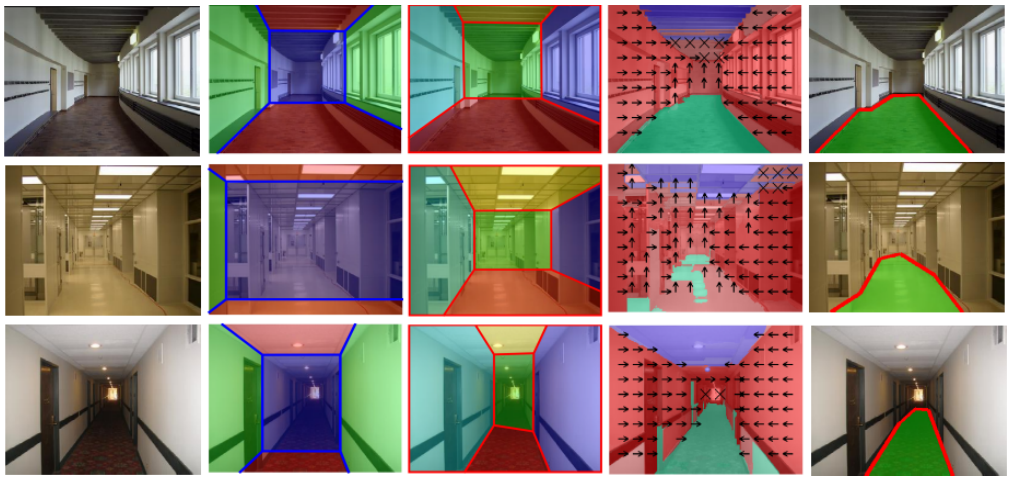


Figure 3.6: Qualitative comparison of corridor scene understanding. **Col 1**: the input image. **Col 2**: building model by Lee *et al.* [76]. Ground region is shown in red. **Col 3**: box layout estimates using Hedau *et al.*'s method [52]. Ground is in red. **Col 4**: surface label prediction by Hoiem et al.'s [78] method. Ground is in cyan. **Col 5**: our algorithm. Red lines are fitted line segments of ground boundaries.

ground polygon, namely the re-projected label from 3D cloud to image pixels. All timings are measured on a desktop CPU (Intel i7, 4.0 GHz) and GPU (for CNN). The algorithm takes about $0.07\,\mathrm{s}$ which could run at $15\,\mathrm{Hz}$, or at $30\,\mathrm{Hz}$ if CRF refinement is omitted. In our mixture dataset, the CNN prediction achieves over 95% pixel accuracy of the ground-wall segmentation and the CRF further improves mean IU by 1.5%. The CRF also has a beneficial effect when using the Pop-up model, increasing mean IU by 2%.

We also observed that the accuracy after pop-up slightly decreased. This is often due to excessive simplification of segmentation boundaries when using line extraction in the pop-up process. However, we need this line simplification in order to pop up a 3D plane world for robot navigation. There is a trade-off between getting higher segmentation accuracy and building a simplified world model.

Table 3.1: Evaluation of each step of our method on our dataset

Name	Pixel accuracy(%)	Mean IU (%)	F.W. IU (%)	Test time(s)
CNN 320×240	96.42	87.10	93.43	0.031
CRF 160×120	96.83	88.69	94.20	0.037
Pop-up (No CRF)	95.19	84.86	91.80	0.003
Pop-up (With CRF)	96.16	86.97	93.07	0.003

A quantitative comparison against other models is shown in Table 3.2. Since other methods may predict the wall or ceiling part, we only evaluate "ground" and "non-ground" labels for fair comparison. We use the publicly available implementation of these methods, in Matlab and C++. Since Hoiem *et al.*[78] also combined geometry modelling with learning, we retrain the surface classifier in [78] using our training dataset and show its results in the last row. The success rate is defined as the percentage of images which could generate valid 3D models by certain methods. Since our method, as well as [78], doesn't make assumptions on the specific environment model or view points, it is more robust than room layout parameterization [52, 76], which might not accurately detect the valid vanishing points or enough lines segments to form feasible room models. In all, our method performs much better than others in terms of accuracy, speed and robustness.

3.4.2 Evaluation on Michigan-Milan Indoor dataset

To demonstrate the generalization, we directly test on this dataset without training or parameters tuning. We evaluate three corridor-similar scenes in this dataset: Corridor, Entrance 1 and 2. Qualitative examples and pop-up models using the provided camera parameters are shown in Figure 3.7. Our method generates good 3D models even in poor lighting and occluded environments. Due to space constraints, we only report the F.W. IU evaluation result in Table 3.3. The trend of other metrics is similar.

From the table, we can clearly see that our method outperforms others in the first two scenes. In the third scene, a very structured Manhattan environment with clear boundaries, the Building

Table 3.2: Evaluation comparison on our validation dataset

Name	Pixel accu(%)	Mean IU (%)	F.W. IU (%)	Test time(s)	Success rate(%)
Our method	**96.16**	**86.97**	**93.07**	**0.071**	**100**
Lee [76]	88.31	68.38	80.49	8.403	88.3
Hedau [52]	88.04	69.60	80.65	17.45	85.8
Hoiem [78]	87.96	70.12	81.12	1.355	**100**
Hoiem*[78]	92.09	75.44	86.32	1.355	**100**

* Retrained on our mixture dataset.

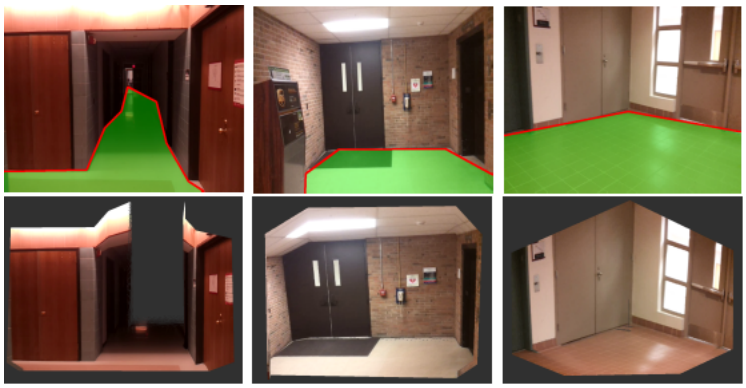


Figure 3.7: Michigan-Milan dataset pop-up examples using our method. The first row is ground segmentation and line fitting. The second row is pop-up 3D model. Scenarios from left to right: Corridor, Entrance 1 and Entrance 2.

model collections of Lee *et al.* [76] outperforms our method by a small margin.

3.5 Analysis

In this section, we analyse how the CNN learns and some of our architectural choices, which contribute most to our segmentation accuracy.Some limitations of the algorithm are also discussed.

3.5.1 What is the CNN learning?

We first visualize some first-layer filters of the CNN in the top row of Figure 3.8. The edges and corners filters in various orientations are important cues to extract and reason about the geometric structure. To visualize what the higher layers of the CNN learn, we retrieve images that maximally activate neurons in these layers. This gives us an understanding of what the neuron is "looking for" in its receptive field [93].

We only select four neurons from `pool5` layer due to space constraints, and for each neuron, we display the top three activation images as shown in the bottom two rows of Figure 3.8. We

Table 3.3: Evaluation comparison on Michigan-Milan (F.W. IU %)

Name	Corridor	Entrance 1	Entrance 2
Our method	**96.66**	**91.17**	97.25
Lee [76]	79.99	80.54	**97.80**
Hedau [52]	87.39	90.70	92.94
Hoiem [78]	78.90	87.71	88.35

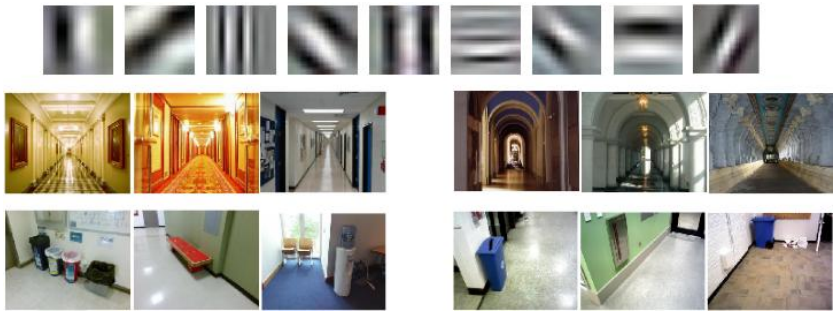


Figure 3.8: CNN visualization. Top row: selected filters from first layer. Bottom two rows: top three activation images of selected four neurons in layer `pool5`. Each set represents certain corridor configurations such as straight forward corridors and left turning corridors.

can see that each set of them represents certain corridor configurations such as long straight corridors, arched ceilings, and dominant left- and right- facing walls.

3.5.2 Why a multiscale CNN?

We use a three-scale CNN to capture both global and local information. Deconvolution layers increase not only the output image resolution, but also segmentation accuracy. We evaluate the contribution of different scales shown in the first two rows in Table 3.4. By adding the third scale, the F.W. IU increases by nearly 3%.

We also compare our CNN model with DSN by Eigen *et al.*[75] shown in the last row of Table 3.4. The main difference is that we use multi-layer summation instead of concatenation in DSN. Our model outperforms DSN model in terms of segmentation accuracy.

Table 3.4: Comparison of CNN different scales and models

Name	Pixel accu(%)	Mean IU (%)	F.W. IU (%)	Output size
Scale 1+2	94.71	81.52	90.15	40×30
Scale 1+2+3	**96.16**	**86.97**	**93.07**	80×60
CNN [75]	95.58	85.15	92.25	147×109

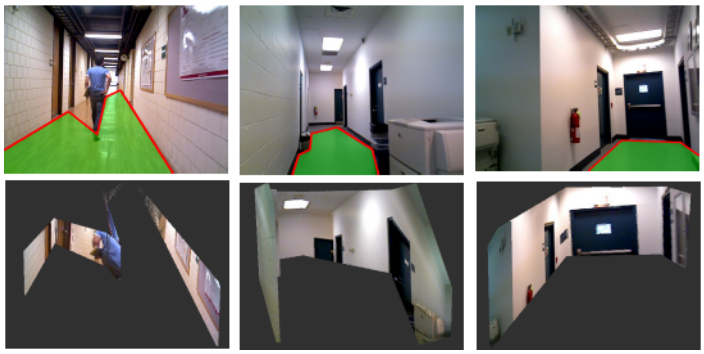


Figure 3.9: Pop-up model in a cluttered environment. Left two columns: person and printer are wrongly popped as wall. Right column: the front-facing wall in left region is wrongly popped as right-facing.

3.5.3 Limitations and future work

Figure 3.9 shows some pop-up models in cluttered environments. Our algorithm can roughly detect the correct ground region, but the 3D model doesn't exactly match the scene geometry due to the following reasons. First, we only model ground and wall so the cluttered objects such as persons, chairs, printers, may be wrongly popped as wall planes in the left two images of Figure 3.9. An extra object detection step could be used to correctly pop these objects. Second, the wall's normal direction is computed based on the corresponding wall-ground boundary. If the boundary cannot be seen or detected correctly, the 3D model may not match the true geometry. For example, in the third image of Figure 3.9, the front facing wall in the left region is mistakenly popped as a right-facing wall. This is also a challenging problem for many existing methods [76, 78]. One possible solution is to separately model the wall and ground planes so that walls can still be popped without a visible ground plane.

3.6 Conclusions

In this chapter, we have presented a system for reliable real-time corridor layout understanding from a single image, which is applicable for robot navigation. The key components of our method are an efficient and accurate CNN+CRF classifier to segment images into two geometric classes, and a pop-up algorithm that uses geometric constraints to create a simplified 3D model. We collect a large dataset of various corridors with nearly 1000 images, and use it to evaluate our method and other state-of-the-art algorithms for this task. We show that our method outperforms other systems in accuracy while labeling frames at real-time rates.

In the future, we are interested in using multiple images in videos to refine the 3D model and obtain more accurate state estimation. This could allow us to build a consistent 3D map. We also would like to improve the modelling of cluttered objects and wall planes to generate a more accurate and complete scene interpretation. We will also test our algorithm on real robots.

Chapter 4

Image based Joint Object and Plane Detection

This chapter presents a method for joint 3D object and plane detection from single image. We first propose an efficient method to generate high quality cuboid proposals for object detection combing deep learning with projective geometry. Then a high order graphical model with occlusion and intersection constraints is formed to select the best plane and object proposals. It is an extension to only layout plane detection in the previous Chapter 3.

4.1 Introduction

Object detection and scene understanding has been widely used in many applications. For example, in autonomous driving, vehicles need to be detected in 3D space in order to remain safe. In augmented reality, 3D objects and layout planes need to be localized for more realistic physical interactions. Different sensors can be used for these tasks such as laser-range finders, stereo or RGB-D cameras which can directly provide depth measurement. For object detection, many algorithms are able to detect different 2D objects with various size and viewpoints in large datasets using convolutional neural network (CNN) [35]. 3D object objection is more challenging and has also attracted attention recently such as vehicle detection [37] [38]. Many of them depend on the object shape priors or fixed object dimensions which limit the application for general environments and objects.

In this work, we propose an algorithm for general 3D object detection and layout understanding. Given the 2D object detection, many 3D cuboid proposals are efficiently generated through vanishing point (VP) sampling, under the assumptions that the cuboid will fit 2D bounding box tightly after projections. The layout plane proposals are also generated based on ground edges then the best subset of object and plane proposals is selected to minimize occlusions and intersections as shown in Fig 4.1. In summary, our contributions are as follows:

- An efficient, accurate and robust single image 3D cuboid detection approach without object shape priors.
- A high order graphical model with efficient inference for the joint structured reasoning of 3D objects and layout planes.

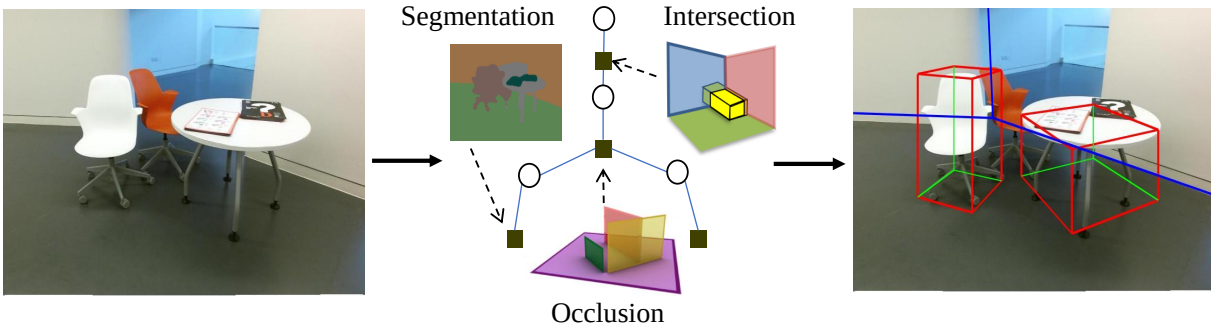


Figure 4.1: Overview of single image 3D object and layout detection. We first efficiently generate many high quality object and layout proposals then formulate a graphical model to select the optimal subset based on evidence of semantic segmentation, intersections, occlusions, and so on.

In the following, we first introduce the related work then present the main parts of 3D object detection and joint object with plane understanding in Sec 4.2 and Sec 4.3. Extensive experiments are carried out in Sec 4.5, followed by conclusions in Sec 4.6.

Closely Related Work

We here briefly review single image object and plane detection. A more complete review about them can be found in Chapter 2. We also review the CRF inference which we contribute to.

Image based 3D object detection Existing 3D object detection approaches can be divided into two categories, with or without shape priors. With prior models, the best object pose to align with RGB images can be found through keypoint Perspective n-Point (PnP) matching [43], hand-crafted texture features [44] or more recent deep networks [38] [45]. Without prior models, the typical approach is to combine geometry modelling with learning. For example, objects can be generated by a combination of Manhattan edges or rays through VPs [94] [46]. Chen *et al.* proposed to exhaustively sample many 3D boxes on the ground then select them based on various context features [49]. One similar work to us is [50][51] which used projective geometry to find cuboids to fit tightly with the 2D bounding box. We extend it to work without prediction of object size and orientation.

Joint object and plane detection For layout plane detection, the popular room model based on vanishing points is proposed by Hedau *et al*[52]. Recent learning based approaches such as LayoutNet [54] and RoomNet [55] can achieve impressive results in Manhattan room environments. These approaches can generate roughly correct plane models, but they are not suitable for SLAM landmark optimization because CNN prediction may be inconsistent across frames. In addition, most of them only apply to the restricted four-wall room models. Our work is more related to the joint and holistic 3D understanding of object and planes. Their positions are optimized based on the spatial and semantic relationships such as occlusion, intersection, and concurrence [56]. Most of them utilize RGBD camera and are not running in real time. More recent works directly predict the 3D occupancy of objects and planes utilizing CNN [59].

Sparse CRF inference There has been lots of research on high order discrete CRFs for computer vision problems [95], however, efficient MAP inference is still challenging in many cases. Markov Chain Monte Carlo (MCMC) sampling is a straight forward method and has already been used in some scene understanding work [96] [97] but it is computationally expensive. Some works utilizes relaxation method to transform the binary CRF into Mixed Integer Linear Program (MILP) [98] [99]. The sparsity of potentials has also been exploited in many inference methods. Rother *et al.* proposed a compact representation to convert sparse potentials into pairwise ones by introducing a small set of auxiliary variables [100]. [101] proposed an efficient message-passing BP algorithm. Some special forms of high order potentials are also explored in [102] [103]. More recently, Geiger *et al.* utilized recursive space-partitioning to handle pattern-based potentials [57], which is a generalization of [101] [100]. Our algorithm is inspired by it but has different potential patterns compared to it.

4.2 3D Object Detection

4.2.1 3D proposal generation

Principles

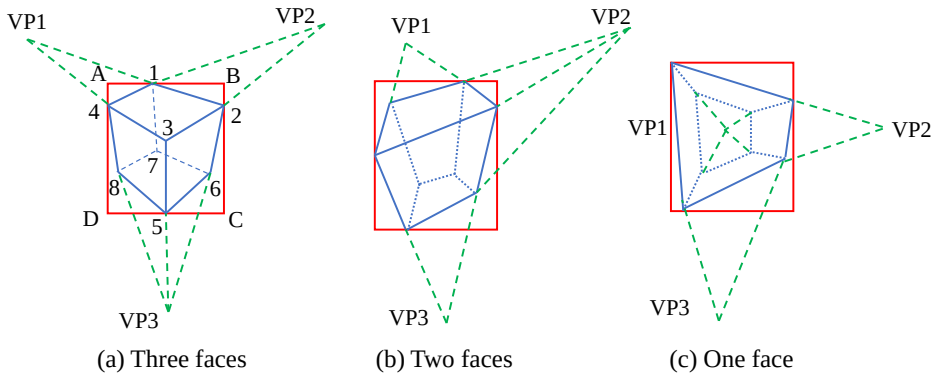
Instead of randomly sampling object proposals in 3D space, we utilize the 2D bounding box to efficiently generate 3D cuboid proposals. A general 3D cuboid can be represented by 9 DoF parameter: 3 DoF position, 3 DoF rotation and 3 DoF dimension. The cuboid coordinate frame is built at the cuboid center, aligned with the main axes. The camera intrinsic calibration K is also known. Based on the assumptions that the cuboid's projected corners should fit tightly with the 2D bounding box, there are four constraints corresponding to four sides of the 2D box which cannot fully constrain all 9 parameters. Thus other information is needed for example the provided or predicted object dimensions and orientations, used in vehicle detection [49] [50] [51]. Different from them, we utilize the VPs to change and reduce the regression parameters in order to work for general objects.

As we know, VP is the parallel lines' intersection after projection onto perspective images. A 3D cuboid has three orthogonal axes thus can form three VPs after projections depending on object rotation R *wrt.* camera frame and the camera intrinsic calibration K:

$$\text{VP}_i = KR_{col(i)}, \quad i \in \{1, 2, 3\} \tag{4.1}$$

where $R_{col(i)}$ is the i^{th} column of the rotation matrix R.

Therefore we can first estimate the object's 3 DoF rotation R to compute three VPs and further estimate one cuboid corner on the image, then all the other seven corners can be computed analytically as shown in Fig. 4.2. After we get 8 cuboid corners in 2D, we can use then PnP solver to compute the cuboid's 3D position and dimensions analytically, but it is up to a scale factor due to monocular scale ambiguity.



(a) Three faces (b) Two faces (c) One face

Figure 4.2: Cuboid proposals generation from 2D object box. If vanishing points (VP) and one corner are estimated, the other seven corners can also be computed analytically. For example in (a), given corner 1, corner 2 and 4 can be determined through line intersection, and same as other corners.

Example

Since at most three cuboid faces can be observed simultaneously, we can divide the cuboid configurations into three categories based on the number of observable faces shown in Fig. 4.2. Each configuration can be left-right symmetric. We explain configuration Fig. 4.2(a) in more details. Suppose three VPs and top corner 1 are known or estimated and $\times$ represents line intersection, then $p_2 = (vp_1, p_1) \times (B, C)$, similar for p_4. Then $p_3 = (vp_1, p_4) \times (vp_2, p_2), p_5 = (vp_3, p_3) \times (C, D)$, similar for the remaining corners.

Ground objects

For many robotic applications with video data, we can change the coordinate frame to make it suitable for latter multi-view optimization and also get the scale factor more easily. We can assume objects lie on the 'ground' surface and build the world frame on the ground. Then camera's roll/pitch angle, and object's yaw are needed in order to compute the relative rotation R between objects and cameras. After the same process as before to compute 8 image corners, we can directly project them onto the 3D ground to formulate a cuboid instead of using PnP solver. The scale is determined by the camera height, which can be provided by SLAM estimation or other sensors.

More importantly, our approach doesn't need to directly predict object full 3D dimensions and orientations through networks [49] [50] [51], thus it is more suitable for detecting diverse objects in various robots application.

Sampling

The problem now changes to how to get object rotation R and one cuboid corner p. Though deep networks can be used to directly predict them with large amounts of data training, we choose to sample manually based on the provided or SLAM estimated camera poses. For a ground object,

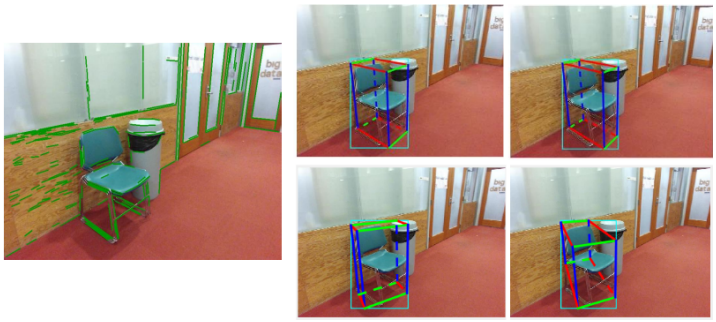


Figure 4.3: Cuboid proposal scoring. (**Left**) Edges to align and score the proposals. (**Right**) Cuboid proposals generated from the same 2D cyan bounding box. The top left is the best and bottom right is the worst after scoring.

as explained before, camera roll/pitch is approximately correct thus the sampling space is greatly reduced and only object yaw needs more samples. On some datasets such as KITTI, the sampling space can be further reduced as the camera is nearly parallel to ground with zero roll/pitch and fixed camera height.

4.2.2 Proposal scoring

After sampling many cuboid proposals, we define cost functions to score them shown in Fig. 4.3. Different cues have been proposed such as semantic segmentation [49], edge distance [44], HOG features [94]. We propose some fast and effective cost functions to align the cuboid with image edge segments. If the image is denoted as I, the fitting error of proposal x is defined as:

$$E(x|I) = \phi_{dist}(x) + w_1\phi_{angle}(x) + w_2\phi_{shape}(x) \tag{4.2}$$

where $\phi_{dist}, \phi_{angle}, \phi_{shape}$ are three costs explained as follows. w_1 and w_2 are weight parameters and set to be $w_1 = 0.7, w_2 = 2.5$ after manual search on small sample datasets.

Distance error ϕ_{dist}

The 2D cuboid edges should match with the actual image edges. We build distance map based on Canny edge detection, then sum over the cuboid edge's Chamfer distance, normalized by 2D box size. Basically, we sample 10 points on each visible cuboid edge then sum over their distance map value.

Angle alignment error ϕ_{angle}

The distance error is sensitive to the noisy false positive edges such as object surface textures. Therefore, we also detect long line segments and measure whether they align with VP computed from cuboid orientations in Sec 4.2.1. The detected lines are first grouped into three VPs based on the point-line support relationship [46]. Then each VP, we select two outmost lines then compute their angle alignment error:

$$\phi_{angle} = \|\theta(a,b) - \theta(vp,b)\| \tag{4.3}$$

where vp is a VP and (a,b) is a detected line segment. θ is the line angle of two points.

Shape error ϕ_{shape}

The previous two costs can be evaluated efficiently just in 2D image space. However, similar 2D cuboid corners might generate quite different 3D cuboids. We add a cost to penalize the cuboids with a large skew ratio (s=length/width). More strict priors could be applied for example the estimated or fixed dimensions of vehicles, chairs and so on.

$$\phi_{shape} = \max(s - \sigma, 0) \tag{4.4}$$

where σ is a threshold. If $s < \sigma$, no penalty is applied.

4.3 Joint Object and Plane Detection

We represent the environment as a set of layout planes such as wall, floor, and cuboid objects similar to some existing work [56]. The goal is to simultaneously infer their locations from 2D image. We first generate a number of object and plane proposals (hypothesis), then select the best subset of them satisfying occlusion constraints via Conditional Random Field (CRF) optimization, as shown in Fig 4.1.

4.3.1 Proposal generation

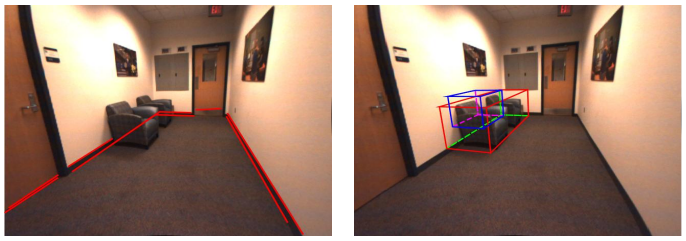
Layout Plane Proposal

We project the actual detected ground-wall edges to 3D space to generate plane proposals, which can be directly used as the latter SLAM landmark because edge observation is consistent across frames. Our prior work [33] also adopts this idea and we extend it to work robustly in large environments with objects.

We first detect all image edges then select some edges close to the ground-wall segmentation [104] boundary. For room environments, layout plane prediction score [53] is additionally used to select possible edges. If the edge lies partially inside object regions, we further extend it to intersect with other edges as shown in Fig 4.4(a), because it may be occluded by foreground objects.

Object Cuboid Proposal

The cuboid proposals are generated in the previous Section 4.2. For each object instance, we select the best 15 cuboid proposals for latter CRF optimization. More cuboid proposals may improve the final performance but also increase the computation a lot. Two of these are shown in Fig 4.4(b) for illustration.



(a) (b)

Figure 4.4: Single image generated plane and cuboid proposals. (a) Wall plane proposals are represented as ground edges. (b) Different cuboid proposals for the same object instance.

4.3.2 CRF Model definition

Given all the proposals, we want to select the best subset from them. We start by defining a binary random variable $x_i \in \{0, 1\}$ for each of the plane and cuboid proposal, indicating whether it will be selected. We use conditional random field (CRF) to model different constraints between them such as intersection and occlusions. The CRF model is shown in Fig 4.5.

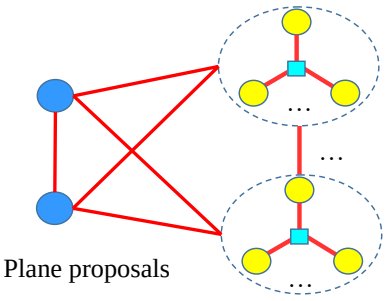


Figure 4.5: The CRF model of single image object and layout plane detection. The blue and yellow circles represent the plane and object proposals respectively. The large dotted circle indicates that all the cuboid proposals inside it belong to one object. The edges and cyan squares represent the relationship constraints between them.

Mathematically, we want to optimize the labels to minimize the following different energy functions or called potentials:

$$E(\mathbf{x}|\mathbf{I}) = \sum_{i} \psi_i^U(x_i) + \sum_{i<j} \psi_{ij}^P(x_i, x_j) + \sum_{\mathbf{x_c}<\mathcal{C}} \psi_{\mathbf{c}}^{HO}(\mathbf{x_c}) \tag{4.5}$$

where ψ_i^U and ψ_{ij}^P are the unary and pairwise potential energy. $\psi_{\mathbf{c}}^{HO}$ is the high order term of clique $\mathbf{x_c}$. These potentials will be explained in more details in the following.

Unary potential

It represents the proposal quality itself. For planes, the energy depends on the edge's distance to the ground-wall segmentation contour and layout edge prediction score if in room environments. Long edges are also preferred compared to short ones which are likely to be outliers due to detection errors. Proper weighting and normalization is needed to combine them together. For objects, we can directly use the cuboid fitting error in the previous Section 4.2, based on the vanishing point and edge alignment. For each proposal, if it appears namely $x_i = 1$, we assign negative unary energy to encourage it.

Pairwise Potential

There are different forms of pairwise relationship between objects and planes for example the semantic co-occurrence [56]. Here we only utilize the geometric relationship to minimize the 3D occlusion and intersection.

For object-object, $\psi^P_{\mathcal{O}-\mathcal{O}}$ is defined as the 3D intersection of union. For object-plane $\psi^P_{\mathcal{O}-\mathcal{L}}$, it represents the truncation ratio of object volume by plane. For plane-plane, $\psi^P_{\mathcal{L}-\mathcal{L}}$ indicates the angle overlapping ratio between each other. Since large plane occlusion is strongly discouraged, if their overlapping angle is greater than 5°, a very large penalty cost is assigned. Note that there is no pairwise potential between cuboid proposals belonging to the same object. For every two proposal, if they appear namely $x_i, j = 1$ and they can intersect or occlude, we assign positive pairwise energy to between them to discourage the intersection or occlusion between them.

$$\psi^P_{\mathcal{O}-\mathcal{O}}(x_i, x_j) = x_i x_j \frac{V(x_i) \bigcap V(x_j)}{V(x_i) \bigcup V(x_j)} \tag{4.6}$$

$$\psi^P_{\mathcal{O}-\mathcal{L}}(x_i, x_j) = x_i x_j \frac{V^{occ}(x_i)}{V(x_i)} \tag{4.7}$$

Where $V(x)$ denotes the space of objects occupied by 3D cuboids. It can also denote the wall planes' angle range when computing plane-plane potential.

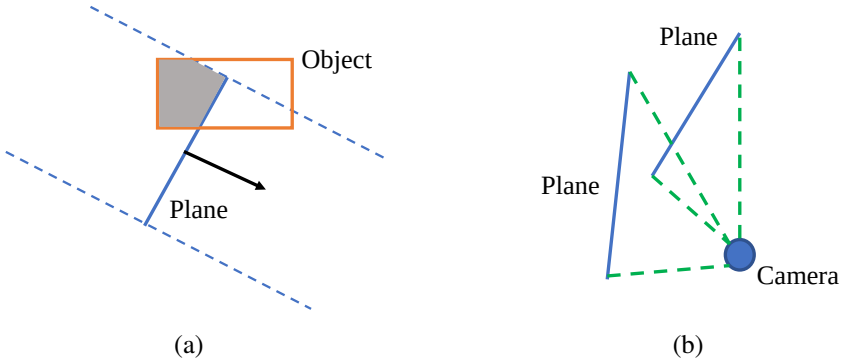
High order potential

As explained in Section 4.3.1, for each 2D object instance, many 3D cuboid proposals are generated from it but at most one of them can be selected. Thus the high order potential becomes:

$$\varphi^{HO}(\mathbf{x_c}) = \begin{cases} 0 & \text{if } \sum_{x_i \in \mathbf{x_c}} x_i \leq 1 \\ \infty & \text{otherwise} \end{cases} \tag{4.8}$$

4.3.3 Efficient CRF inference

There has been lots of research on high order discrete CRFs [95] but efficient inference is still challenging in many cases. However, our high order term in Eq 4.8 is very sparse because in



(a) (b)

Figure 4.6: (a) Object-layout plane intersections (top view). The object is being occluded by the scene plane. The ratio of gray volume is defined as the potential. (b) Layout-layout occlusions (top view). The two planes have angle overlapping and occlusion with each other. The potential is defined as the ratio of overlapped angle by total angle.

one clique $\mathbf{x_c}$, at most one variable can be 1, thus we can design efficient inference for it. Max-product loopy belief propagation [105] is adopted. The messages between variable node i and factor of clique c are passed as follows:

$$m^t_{c\to i}(x_i) = \min_{\mathbf{x}_c^{-i}} \left(f_c(\mathbf{x}_c) + \sum_{j\in c\setminus\{i\}} m^{t-1}_{j\to c}(x_j) \right) \tag{4.9}$$

$$m^t_{i\to c}(x_i) = \sum_{c'\in\mathcal{N}_i\setminus\{c\}} m^{t-1}_{c'\to i}(x_j) \tag{4.10}$$

where $\mathbf{x}_c^{-i}$ denotes all the variables in clique c except variable i, $m^t_{i\to c}(x_i)$ denotes the message from variable to clique, which is easy to compute. The main bottleneck is the message from clique to variable $m^t_{c\to i}(x_i)$ because $\min_{\mathbf{x}_c^{-i}}$ evaluates all states of a clique except node i. For a clique with N binary nodes, there are totally 2^N clique states. However there are only $N+1$ valid states in our problem $\{1,0,...0\},...\{0,0,...1\},\{0,0,...0\}$ denoted as $\{\mathbf{y}_1,\mathbf{y}_2,...\mathbf{y}_{N+1}\}$. Therefore, we only need to check $N+1$ states and find the minimum in Eq 4.9. We can further observe that every adjacent $\mathbf{y}_i$ only has two different variables, therefore $\sum_{j\in c\setminus\{i\}} m^{t-1}_{j\to c}(x_j)$ for each $\mathbf{y}_i$ can be computed iteratively. Therefore, the average time complexity of computing $m^t_{c\to i}(x_i)$ is $O(1)$ instead of the naive $O(2^N)$. More details can be found at the appendix.

4.4 Implementation

Object detection For the 2D object detection, YOLO detector [36] with probability threshold of 0.25 is used for indoor scenarios and MS-CNN [106] with probability of 0.5 is used for outdoor KITTI. Both can run in real time on GPU.

If accurate camera pose is known for example in SUN RGBD dataset, we only need to sample object yaw to compute VPs as explained in Section 4.2.1. We need to note that, without object

shape priors, our cuboid detection cannot differentiate between the front or back face of objects. For example, we can represent the same cuboid by rotating the object coordinate frame by 90 and swapping length with width value. Therefore, we only need to sample object yaw in a range of $90°$. 15 samples generated inside this range. Then 10 points are sampled on the top edge of the 2D bounding box. Note that not all the samples can form valid cuboid proposals as some cuboid corners might lie outside of the 2D box. In scenarios with no ground truth camera pose provided, we sample camera roll/pitch as well in the feasible range or based on SLAM pose estimation. One advantage of our approach is that it doesn't require large training data as we only need to tune the two cost weights in Eq 4.2. It can also run in real time including 2D object detection and edge detection.

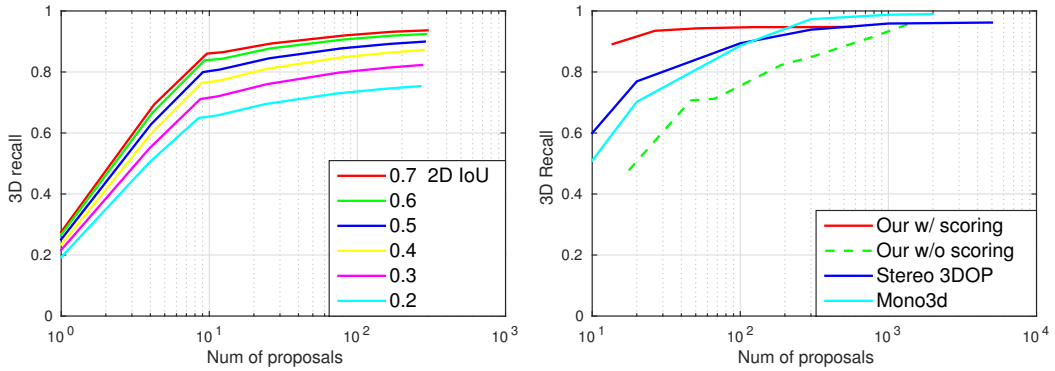
Joint Detection For plane proposals, we first detect and merge broken and noisy line segments. Then the lines shorter than 50 pixels and more than 50 pixels away from the wall-ground segmentation boundary are deleted. If camera rotation especially roll/pitch is roughly correct, we also back-project lines segments to the 3D ground to remove some very far away lines or lines behind the camera. As mentioned before, we need to extend edges occluded by objects. The edges whose endpoints are within 50 pixels from the object boundary are selected as potential edges. Then every pair of them can generate possible intersection and we check whether it lies in object region and doesn't increase the line segments too much. These strict tests can reduce many false positive extensions. Even the occluded lines are not extended, it won't have large effect on the final 3D understanding.

We also need to assign weights for different CRF energy functions in Eq 7.3. We manually search the parameters. For different pairwise potentials, they all represent the intersection or occlusion ration between 0-1, therefore, we assign similar weight parameters between them. As mentioned before, unary potential is computed by normalizing the cuboid fitting or layout score, therefore, we need to manually scale them so that the final unary is in the same magnitude of pairwise energy.

4.5 Experiments

4.5.1 3D Object Detection Result

SUN RGBD [3] and KITTI [107] dataset with ground truth 3D bounding box annotations are used for single view object detection evaluation. 3D intersection over union (IoU) is adopted as the evaluation metric instead of only rotation or viewpoint evaluation in many other works. If the 3D IoU is greater than 25%, the cuboid is treated as a positive detection [3] [49]. Since our approach doesn't depend on the prior object model, in order to get an absolute scale of object position and dimensions, we only evaluate the ground objects with known camera height as explained in Sec 4.2.1. For the KITTI dataset, this assumption is already satisfied. For the SUN RGBD dataset, we select 1670 images with visible ground plane and ground objects fully in the field of view.



(a) (b)

Figure 4.7: (a) Proposal recall on SUN RGBD Subset dataset. Different lines correspond to different 2D box IoU. (b) Proposal recall on KITTI dataset. Our approach can get high recall with low proposals.

Proposal Recall

We first evaluate the quality of proposal generation in SUN RGBD. It is obvious that if the 2D bounding box is inaccurate, our 3D cuboid accuracy will also be affected. We study this effect by evaluating objects only greater than certain 2D IoU threshold τ shown as different curves in Fig. 4.7(a). Obviously, larger τ indicates higher 3D recall. Our approach can achieve 3D recall of 90% using around 50 cuboid proposals when 2D IoU is 0.6.

We then evaluate and compare the proposal quality on the KITTI dataset shown in in Fig. 4.7(b) using the training and validation split by [37]. Note that [49] (Mono3d) first exhaustively samples huge amounts cuboid proposals ($\sim$14k) then reports the recall after selecting the top N proposals based on semantic segmentation and so on thus we also evaluate the recall before and after scoring. Before scoring (green line), our approach can reach a recall of 90% with 800 raw proposals per image, about 200 proposals per object. After scoring (red line), we can reach the same recall using just 20 proposals, much less compared to [49]. There are two main reasons for that: one is that our 3D proposals are of high quality because they are guaranteed to match the 2D detected box. Another reason is the more effective scoring function. Note our approach has an upper limit shown in Fig. 4.7(b) because 2D detector might miss some objects.

Final detection

We then evaluate the final accuracy of the best selected proposal. In SUN RGBD, to our knowledge, we didn't find the trained 3D detection algorithm on it. We thus compare with two public approaches SUN primitive [94] and 3D Geometric Phrases (3dgp) [108] which are both model based algorithms similar to us. 3dgp additionally uses fixed prior object models. We modify their code to use the actual camera pose and calibration matrix when detecting and unprojecting to 3D space. To eliminate the effect of 2D detector, for all the methods, we only evaluate 3D IoU for objects with 2D box IoU>0.7. As shown in Table 4.1 and Fig. 4.8, our approach is more robust as it can detect many more accurate cuboids. Our mean 3D IoU is smaller compared 3dgp

Figure 4.8: 3D object detection examples in SUN RGBD and KITTI dataset (Single view).

using prior models but higher if we only evaluate on the same detected objects by 3dgp.

On the KITTI dataset, we compare with two other monocular algorithms [50] [37] using deep networks. From Table 4.1, our approach performs similarly to SubCNN which uses prior models. It performs worse compared to [50] which directly predicts vehicle orientations and dimension. As there is only one object class "car" with fixed camera poses and object shapes, CNN prediction works better than our hand-designed features, also explained in Sec 4.2.1. In the last row, we also include the evaluation of the selected top 10 cuboid proposals to show that our proposals are of high quality even with a few.

More Object Detection Result

We also provide 3D object detection of the two datasets shown in Fig 4.9. The red and blue rectangles show the ground truth and predicted object respectively. We can see that it performs well on front-view boxy objects such as chairs and tables. For the vehicle detection, the orientation is not accurate sometimes due to unobvious edges.

4.5.2 Joint Object and Plane Detection

We first show the single image layout plane and cuboid object detection result. Some examples of proposal generation and CRF optimization are shown in Fig 4.10. The middle and right columns show the top view of object proposals before and after CRF optimization. We can easily see from it that CRF can select non-overlapped wall edges and better cuboid proposals to minimize occlusions and object intersection.

More results of CRF selected object and plane proposals are shown in Fig 4.11. The algorithm is able to work in different environments from rooms to corridors but it may still fail to detect all

Figure 4.9: More 3D object detection examples in SUN RGBD and KITTI evaluation dataset (Single View). The top view shows the predicted cuboid (blue) and ground truth matched object (red).

Table 4.1: Comparison of 3D object detection on SUN RGBD subset and KITTI dataset

	Method	3D IoU	Detected cuboids
SUN RGBD	Primitive[94]	0.36	125
	3dgp[108]	0.42	221
	Ours	0.39	1890
	Ours*	0.45	252
KITTI	Deep[50]	0.33	10957
	SubCNN[37]	0.21	8730
	Ours	0.21	10406
	Ours top 10	0.38	10406

* On 3dgp detected images.

Table 4.2: 3D Object IoU of joint scene understanding on SUN RGBD subset data

Method	Before CRF[109]	After CRF
3D IoU	0.35	0.40

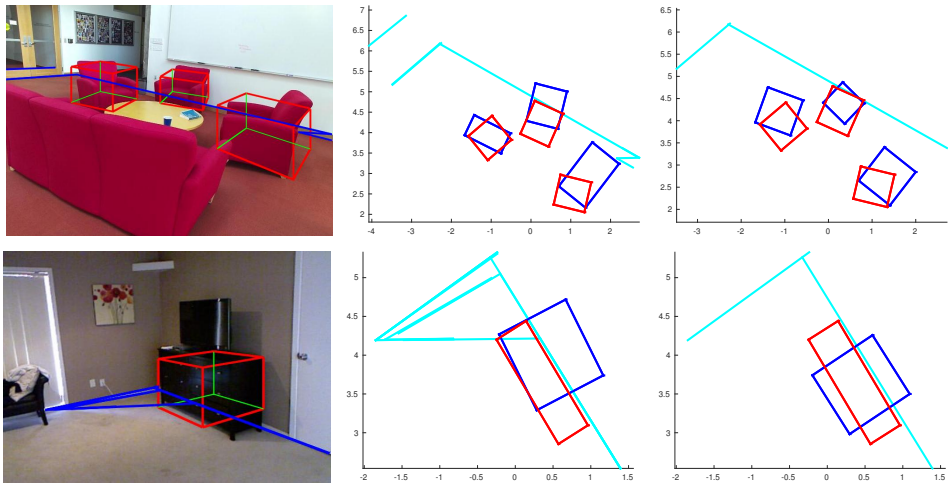
the wall planes and objects when there is severe object occlusion and unclear edges for example in the right column of Fig 4.11.

We also evaluate quantitatively the CRF optimization performance on SUN RGBD dataset. Compared to [109] (in Section 4.2) which selects the best cuboid proposal without considering planes, our CRF joint reasoning of object and plane improves the 3D object intersection over union (IoU) by 5% shown in Table 4.2. Note that to emphasize the optimization effect, we only evaluate on images where CRF generates different results, no matter good or bad, compared to the single image detection [109]. This is because many images have no visible ground edges or they are far from objects and have no actual constraints on object positions thus CRF optimization will have no effects on those object detections.

4.5.3 Discussions

Though our approach shows the robustness without prior shape models, there are still assumptions and limitations about the cuboid detection part.

1. The approach works the best for "boxy with clear edge" objects and doesn't work well for low texture or symmetric objects. As explained in Sec 4.2.2, cuboid proposals are scored based on the alignment with image line segments. Therefore, for a roundtable in Fig 4.12(a), we cannot precisely determine the object yaw due to symmetry. However, due to the constraints from vanishing points, only some proposals can form valid cuboids fitting the 2D box, therefore, the final 3D location is still roughly correct shown in Fig 4.1.
2. Some special object configurations. First, our method doesn't work directly for some large objects exceeding camera height for example the bed in Fig 4.12(b). We need to



(a) (b) (c)

Figure 4.10: Single image raw proposal generation and CRF optimization illustrations. (a) Raw plane and object proposals. (only draw one cuboid for brevity) (b) Top view of raw proposals. Red rectangle is ground truth object and blue is estimated. Cyan line is wall plane edges. (c) Top view of CRF selected proposals. Object pose is more accurate after optimization. Plane and object occlusion is also minimized.

add additional configuration analysis to Fig 4.2. Second, it is not suitable for non-ground objects such as the table lamp shown in Fig 4.12(b), because we then need to sample object full 3D rotation in order to compute vanishing points. It is also difficult to analyze all the corner configurations for arbitrary 3D object pose.

3. Objects should be fully in the image field of view. For a car in Fig 4.12(c), our method may generate very bad result especially for the object size, because it can only find cuboids to fit the cropped 2D box instead of the original box. Therefore, if the 2D bounding box is too close to the boundary, we won't detect 3D cuboid for it.

Actually, even the single detection result is not accurate or fails to detect some objects or planes, the latter multi-view SLAM with data association can further optimize the object and plane position and build a 3D map of them. This is the key motivation of this thesis.

4.6 Conclusions

In this chapter, we propose an algorithm for 3D object detection and layout understanding from a single image. The object detection doesn't depend on the object shape priors and the layout understanding is not limited to the four-wall Manhattan room models. Therefore, our approach can work in more general environments compared to others.

For the single image 3D object detection, we propose a new method to efficiently generate high quality cuboid proposals using vanishing points based on the assumptions that cuboids fit tightly with 2D box after projection. The proposals are then scored efficiently by image edge

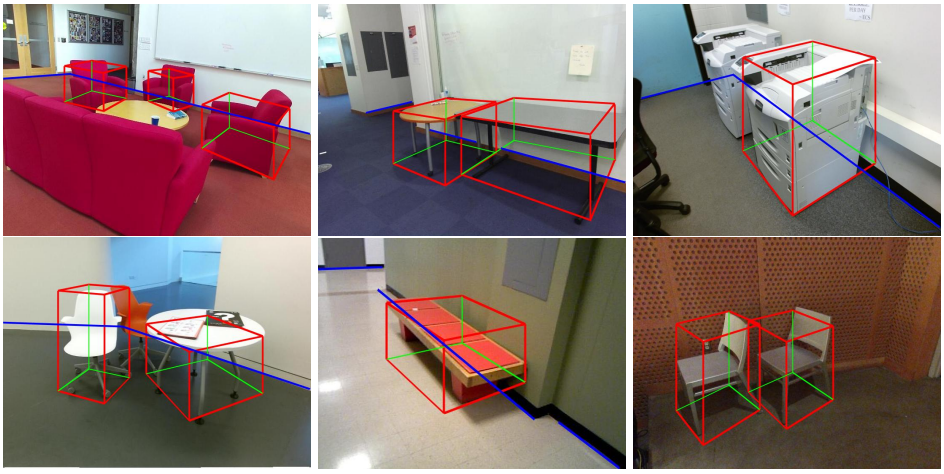


Figure 4.11: More single image CRF optimized object and plane proposals.

(a) (b) (c)

Figure 4.12: Assumptions and limitations of cuboid detection. (a) Round table without long edges is not suitable for cuboid proposal scoring. (b) Some special cases don't belong to the three cube configurations analysed in Fig 4.2, for example the bed in red box. Our method is not suitable for objects not on ground because many more parameters need to be sampled, shown as the lamp in blue box. (c) The method cannot work for objects partially in field of view such as the car in red box

features. Plane proposal are also efficiently generated based on wall-ground edges. We then formulate a high order CRF graphical model to jointly reason the 3D model by minimizing the occlusions and intersections between proposals. A fast inference algorithm for this high order CRF is also proposed.

We evaluate the two parts on indoor SUN RGBD dataset and outdoor KITTI. It achieves the best accuracy of 3D object detection on SUN RGBD subset data and also comparable result on KITTI. The joint CRF understanding can further improve the object detection accuracy.

Chapter 5

Pop-up SLAM: Monocular Plane SLAM

In this chapter, we propose a novel monocular plane SLAM to jointly optimize 3D scene layout planes from multiple image. It is an improvement and extension to single image plane detection in Chapter 3. This chapter mainly works in corridor-like scenarios and more complicated environment with objects are addressed in the next Chapter 6.

5.1 Introduction

SLAM has been widely used for tasks including autonomous navigation, 3D mapping and inspection. Monocular cameras are a popular choice of sensor on robots as they can provide rich visual information at a small size and low cost, which are especially suitable for weight constrained micro aerial vehicles that can carry only one camera.

On one hand, many existing visual SLAM methods utilize point features such as direct LSD SLAM [25] and feature based ORB SLAM [1]. These methods track features or high-gradient pixels across frames to find correspondences and triangulate depth. They usually perform well in environments with rich features but cannot work well in low-texture scenes as often found in corridors. In addition, the map is usually sparse or semi-dense, which does not convey much information for motion planning.

On the other hand, humans can understand the layout, estimate depth and detect obstacles from a single image. Many methods have been proposed to exploit the geometry cues and scene assumption to build simplified 3D models. Especially in recent years, with the advent of Convolutional Neural Networks (CNN) [82] [83], performance of visual understanding has been greatly improved.

In this chapter, we combine scene understanding with traditional v-SLAM to improve the performance of both state estimation, dense mapping and scene understanding especially in low-texture environments. We begin with a single image pop-up plane model [110] (Chapter 3) to generate plane landmark measurements in SLAM. With proper plane association and loop closing, we are able to jointly optimize scene layout and camera poses of multiple frames in the SLAM framework. In the low-texture environment of Figure 5.1, our algorithm can still generate dense 3D models and decent state estimates while other state-of-the-art SLAM fail. However, plane SLAM can easily be under-constrained, hence we propose to combine it with traditional

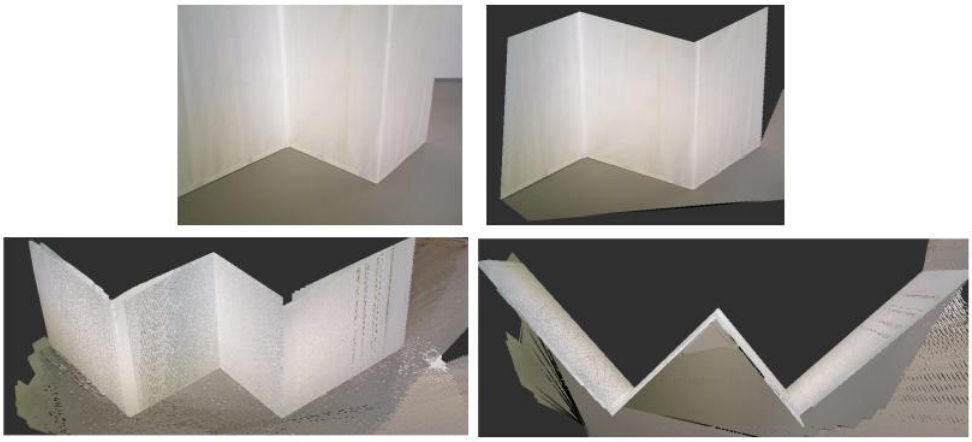


Figure 5.1: 3D reconstruction on low-texture TUM dataset. (Top left) single raw image. (Top right) singe image pop-up 3D plane model. (Bottom) multiple frame's dense reconstruction using our *Pop-up Plane SLAM*. Two views provided. Each plane also has a label of either a specific wall or ground. Existing state-of-art SLAM algorithms fail on this.

point-based LSD SLAM [25] to increase robustness.

In summary, our main contributions are:

- A real-time monocular plane SLAM system incorporating scene layout understanding,
- Integrate planes with point-based SLAM for robustness,
- Outperform existing methods especially in some low-texture environments and demonstrate the practicability on several large datasets with loops.

In the following section, we discuss related work. Section 5.2 describes the single image layout understanding, which provides plane measurements for plane SLAM. In Section 5.3, we introduce the *Pop-up Plane SLAM* formulation and combine it with point SLAM in Section 5.4. Experiments on a public TUM dataset and actual indoor environments are presented in Section 5.5. Finally, we conclude in Section 5.6.

Closely Related Work

Our approach combines aspects of two research areas: single image scene understanding and multiple images visual SLAM. We provide a brief overview of the two areas at the time of publication. A more complete review can be found in Chapter 2.

Single Image Scene understanding There are many methods that attempt to model the world from a single image. Two representative examples are cuboidal room box model based on vanishing points by Hedau *et al*.[52] and fixed building model collections based on line segments by Lee *et al*.[76]. Our previous work [110] (Chapter 3) proposed the pop-up 3D plane model, combining CNNs with geometry modeling. Results show that our work is more robust to various corridor configurations and lighting conditions than existing methods.

Multi-view Visual SLAM Structure from Motion and v-SLAM have been widely used to obtain 3D reconstructions from images [24]. These methods track image features across multiple frames and build a globally consistent 3D map though bundle adjustment optimization. Two representatives of them are direct LSD SLAM [25] and feature- based ORB SLAM [1]. But these methods work poorly in low-texture environments because of the sparse visual and geometric features.

Planes or superpixels have been used in [66, 67, 68] to provide dense mapping in low-texture areas. But they assume camera poses and point cloud are provided from other sources such as point based SLAM, which may not work well in texture-less environments as mentioned above. Recently, Concha *et al*.[9] also proposed to use room layout information to generate depth priors for dense mapping however they don't track and update the room layout thus can only work in small workspace.

Multi-view Scene understanding Some works also address scene understanding using multiple images, especially in a Manhattan world. Flint *et al*.[111] formulated it as Bayesian framework using monocular and 3D features. [6, 79] generated many candidate 3D model hypotheses and subsequently update their probability by feature tracking and point cloud matching. Unfortunately, these methods do not use a plane world to constrain the state estimation and point cloud mapping.

5.2 Single Image Plane Pop-up

This section extends our previous work [110] to create a pop-up 3D plane model from a single image. We first briefly recap the previous work, discuss its limitation, and propose two improvements accordingly.

5.2.1 Pop-up 3D Model

There are three main steps in [110] to generate 3D world: CNN ground segmentation (optionally with Conditional Random Field refinement), polyline fitting, and pop up 3D plane model. It outperforms existing methods in various dataset evaluation. However, there are some limitations:

Firstly, [110] fitted polylines along the detected ground region which might not be the true wall-ground edges and thus generate a invalid 3D scene model. For example in Figure 5.2, it cannot model the right turning hallway. This results in problems attempting to use these planes in SLAM framework because even in adjacent frames, the fitted line segments may be different. However, SLAM requires the landmark (in our case, planes), to be *invariant* across frames.

Secondly, [110] used a zero rotation pose assumption, which in most cases, is not satisfied. Different rotation angles may generate different pop-up 3D model.

In the following two sub-sections, we solve these problems and generate a more accurate 3D map shown at the bottom of Figure 5.2.

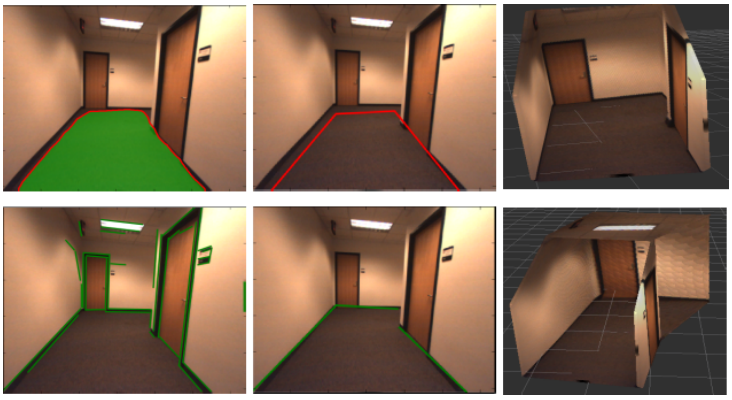


Figure 5.2: Single image pop-up plane model. (Top) original method of [110]. From left to right: CNN segmentation, polyline fitting, pop-up 3D model. (Bottom) improved method. From left to right: line segment detection, selected ground-wall edges, pop-up 3D model. Better ground edge detection and camera pose estimation lead to a more accurate 3D model.

5.2.2 Optimal Boundary Detection

Instead of using a fitted line segments, we detect the true ground-wall edges. We first extract all the line segments using [112], which may has detection noise as as other line detectors. For example, a long straight line may be detected as two disconnected segments. Therefore, we propose an algorithm to optimally select and merge edges as a wall-ground boundary shown in the bottom center of Figure 5.2.

Mathematically, given a set of detected edges $V = \{e_1, e_2, ..., e_n\}$, we want to find the optimal subset edges $S \subseteq V$, such that:

$$\max_{S \subseteq V} F(S),\ st\colon S \in I \tag{5.1}$$

where F is the score function and I is the constraint. Due to the complicated scene structures in the real world, there is no standard way of expressing F and I as far as we know, so we intuitively design them to make it more adaptable to various environments, not limited to a Manhattan world as it is typically done in many current approaches [9] [6].

The first constraint indicates that edges should be close to the CNN detected boundary curve ξ within a threshold shown as red curve in the top left of Figure 5.2. It can be denoted as:

$$I_{close} = \{S\colon \forall e \in S,\ dist(e, \xi) < \delta_{close}\} \tag{5.2}$$

The second constraint is that edges should not overlap with each other beyond a threshold in image horizontal direction shown in Figure 5.3(a). This is true for most cases in the real world. In the latter experiments, we find that even for the unsatisfactory configurations in Figure 5.3(b), our algorithm can select most of the ground edges. We can denote this constraint as:

$$I_{ovlp} = \{S\colon \forall e_i, e_j \in S, O(e_i, e_j) < \delta_{ovlp}\} \tag{5.3}$$

where O is horizontal overlapping length between two edges.

(a) (b)

Figure 5.3: (a) Desired corridor configurations where our algorithm can select all the ground edges. (b) Unsatisfactory configurations because of too much overlap horizontally. Our algorithm might miss some ground edges.

Similarly, we want to maximize the covering of edges in image x direction. So the score function is defined as:

$$F\colon \{S \to \mathbb{R},\ F = C(S)\} \tag{5.4}$$

where C is the horizontal covering length of edge sets S.

With the defined score function F and constraints $I = I_{close} \cap I_{ovlp}$, the problem changes to a submodular set optimization. We adopt a greedy algorithm [113] to select the edges in sequence. We initially start with an empty set of edges S, then iteratively add edges by:

$$S \leftarrow S \cup \{ \underset{e \notin S: S \cup \{e\} \in I}{\arg\max}\ \triangle(e \mid S)\} \tag{5.5}$$

until there is no feasible edges. $\triangle(e \mid S)$ is the marginal gain of adding edge e into set S. The proof of optimality is presented as follows.

After getting the edge set S, some post processing steps are required for example removing tiny edges and merging adjacent edges into a longer one similar to [52].

Submodularity and Optimality Proof

We prove that the aforementioned problem is a submodular set selection problem with matroid constraints [113]. The score function F in Equation (5.4) is obviously monotonically increasing because adding more edges, the covering in image horizontal direction will not decrease.

We first define the marginal gain of e *wrt.* S as the increase of score F after adding element e into S, namely

$$\triangle(e \mid S) := F(S \cup \{e\}) - F(S)$$

For two sets $S_1 \subset S_2$, edge e may overlap with more edges in S_2 and thus reduce the marginal gain compared to S_1, so it satisfies the submodularity condition:

$$\triangle(e \mid S_1) \geq \triangle(e \mid S_2),\ \forall S_1 \subseteq S_2$$

Matroid constraint type We can remove the edges that are far from CNN boundary before submodular optimization, so we only consider the second constraint I_{ovlp} in Equation (5.3). Denote all the conflicting edge pairs as $E_i = \{(e_{i1}, e_{i2}) \mid O(e_{i1}, e_{i2}) \geq \delta_{ovlp}\}, i = 1, 2, ..., k$. For each E_i, we form a partition of the ground set V by two disjoint sets $P_i = \{E_i, V \setminus E_i\}$ and thus can form a partition matroid constraint $I_i^m = \{S\colon |S \cap P_i^1| \leq 1, |S \cap P_i^2| \leq n\}$, where P_i^1 and P_i^2 are two elements of P_i. This is because we can pick at most one element from E_i. The union of k such separate matroid constraints forms the original constraint $I_{ovlp} = I_1^m \cap I_2^m ... \cap I_k^m$.

Optimality From [113], the greedy algorithm in Equation (5.5) of the submodular optimization with matroid constraints is guaranteed to produce a solution S such that $F(S) \geq \frac{1}{k+1} \max_{S \subseteq I} F(S)$. It is also important to note that this is only a worst case bound and in most cases, the quality of solution obtained will be much better than this lower bound.

5.2.3 Pop-up World from an Arbitrary Pose

Notations. We use subscript w to represent global *world* frame and c to denote local *camera* frame. gnd is short for ground plane. A plane can be represented as a homogeneous vector $\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3, \pi_4)^\top = (\mathbf{n}^\top, d)^\top$, where $\mathbf{n}$ is the plane normal vector, and d is its distance to the origin [114] [32]. The camera pose is represented by the 3D Euclidean transformation matrix $\mathrm{T}_{w,c} \in \mathbf{SE}(3)$ from local to global frame. Then a local point $\mathbf{p}_c$ can be transformed to global frame by: $\mathbf{p}_w = T_{w,c}\mathbf{p}_c$, and a local plane $\boldsymbol{\pi}_c$ is transformed to global frame by:

$$\boldsymbol{\pi}_w = \mathrm{T}_{w,c}^{-\top} \boldsymbol{\pi}_c \tag{5.6}$$

Create 3D model

For each image pixel $\mathbf{u} \in \mathbb{R}^3$ (homogeneous form) belonging to a certain local plane $\boldsymbol{\pi}_c$, the corresponding 3D pop-up point $\mathbf{p}_c$ is the intersection of backprojected ray $\mathrm{K}^{-1}\mathbf{u}$ with plane $\boldsymbol{\pi}_c$:

$$\mathbf{p}_c = \frac{-d_c}{\mathbf{n}_c^\top (\mathrm{K}^{-1}\mathbf{u})} \mathrm{K}^{-1}\mathbf{u} \tag{5.7}$$

where $\mathbf{K}$ is calibration matrix.

Then we show how to compute the plane equation $\boldsymbol{\pi}_c$. Our world frame is built on the ground plane represented by $\boldsymbol{\pi}_{gnd,w} = (0, 0, 1, 0)^\top$. Suppose a ground edge's boundary pixels are $\mathbf{u}_0, \mathbf{u}_1$, their 3D point $\mathbf{p}_{c0}, \mathbf{p}_{c1}$ can be computed by Equation (5.6) (5.7). Using the assumption that wall is vertical to the ground, we can compute the wall plane normal by:

$$\mathbf{n}_{wall,c} = \mathbf{n}_{gnd,c} \times (\mathbf{p}_{c1} - \mathbf{p}_{c0}) \tag{5.8}$$

We can further compute $d_{wall,c}$ using the constraints that two points $\mathbf{p}_{c0}, \mathbf{p}_{c1}$ lying on the wall.

Camera pose estimation

The camera pose $\mathrm{T}_{w,c}$ could be provided from other sensors or state estimation methods. Here, we utilize a single image attitude estimation method which could be used at the SLAM initialization stage. For a Manhattan environment, there are three orthogonal dominant directions $\mathbf{e}_1 = (1, 0, 0)^\top, \mathbf{e}_2 = (0, 1, 0)^\top, \mathbf{e}_3 = (0, 0, 1)^\top$ corresponding to three vanishing points $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \in \mathbb{R}^3$ in homogeneous coordinate. If the camera rotation matrix is $\mathbf{R}_{w,c} \in \mathbb{R}^{3\times3}$, then $\mathbf{v}_i$ can be computed by [52] [115]:

$$\mathbf{v}_i = \mathbf{K}\mathbf{R}_{w,c}\mathbf{e}_i, \quad i \in \{1, 2, 3\} \tag{5.9}$$

With three constraints of Equation (5.9), we can recover the 3 DoF rotation $\mathbf{R}_{w,c}$.

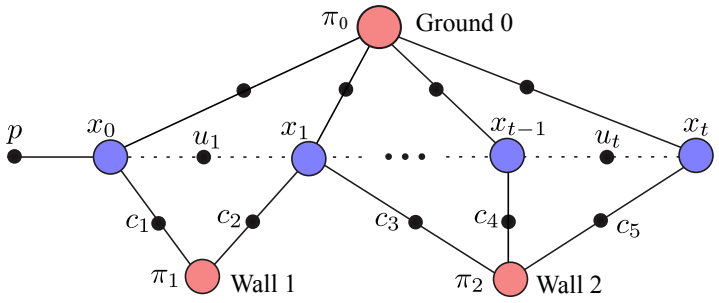


Figure 5.4: Plane SLAM factor graph. Variable nodes include camera pose x, plane landmark π. Factor nodes are odometry measurements u and plane measurements c. The latter come from the single image pop-up model. Each plane node π also has a label of either ground or wall.

5.3 Pop-up Plane Slam

This section introduces the *Pop-up Plane SLAM* using monocular images. Plane SLAM has recently been addressed by Kaess [32] with a RGB-D sensor, here we extend it to the monocular case based on the pop-up plane model in the previous section.

5.3.1 Planar SLAM Formulation

The factor graph of planar SLAM is shown in Figure 5.4. We need to estimate the 6 DoF camera poses $x_0, ..., x_t$ and plane landmarks $\boldsymbol{\pi}_0, ..., \boldsymbol{\pi}_n$ using the plane measurements $c_0, ..., c_m$, odometry measurements $u_1, ..., u_t$ and initial pose constraint p. Note that, our plane landmark also has a label being either ground or wall. The ground plane landmark $\boldsymbol{\pi}_0$ is connected to all pose nodes.

The homogeneous plane representation $\boldsymbol{\pi} = (\mathbf{n}^\top, d)^\top$ is over-parametrized and therefore the information matrix of SLAM is singular and not suitable for Gauss-Newton solver and incremental solvers such as iSAM [16]. We utilize the minimal plane representation in [32] to represent planes as a unit quaternion $\mathbf{q} = (q_1, q_2, q_3, q_4)^\top \in \mathbb{R}^4$ st. $\|q\| = 1$. We can therefore use Lie algebra and exponential map to do plane updates during optimization.

5.3.2 Plane Measurement

Most plane SLAM [32, 116] uses RGB-D sensor to get plane measurements c from the point cloud segmentation. However, in our system, plane measurements c come from the pop-up plane model in Section 5.2.3. Note that the pop-up process depends on the camera pose, more specifically rotation and height because camera horizontal position does not affect local plane measurements. So we need to *re-pop up* the 3D plane model and update plane measurement c after camera poses are optimized by plane SLAM. This step is fast with simple matrix operation explained in Section 5.2.3. It takes less than 1ms to update hundred's plane measurement.

5.3.3 Data Association

We use the following three geometry information for plane matching: the difference between plane normals, plane distance to each other and projection overlapping between planes. The plane's bounding polygon comes from the pop-up process. Outlier matches are first removed by thresholds of the three metrics. Then the best match is selected based on a weighted sum of them shown in Fig 5.5.

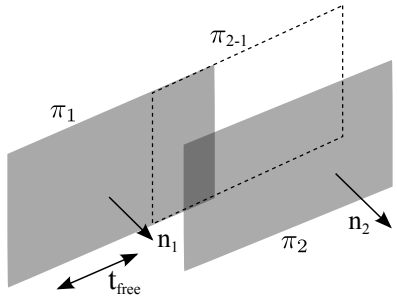


Figure 5.5: Data association and unconstrained plane configurations. π_1, π_2 are two planes with normals n_1, n_2 respectively. π_{2_1} is the projected plane from π_2 onto π_1. Overlapping between π_{2_1} and π_1 is used for data association. In this example, n_1 and n_2 are parallel so there is an unconstrained direction along t_{free}.

5.3.4 Loop Closure

We adopt a bag of words (BoW) place recognition method [117] for loop detection. Each frame is represented as a vector of visual worlds computed by ORB descriptors so as to calculate the similarity score between two frames. Once a loop closure frame is detected, we search all the plane pairs in the two frames and find the plane pairs with smallest image space distance. We also tested to keep BoW visual words for each plane, but it is not robust especially in texture-less images. Different from point landmarks, plane landmarks have different appearance and size in different views. So we may recognize the same planes after the landmark has been created and observed for sometime. So after detecting, for example, π_n and π_2 as the same plane in Figure 5.6, we *shift* all the factors of plane π_n to the other plane π_2, and remove the landmark π_n from factor graph.

5.4 Point-Plane SLAM Fusion

Compared to point based SLAM, planar SLAM usually contains much fewer landmarks so it becomes easily unconstrained. For example in a long corridor in Figure 5.5 where left and right walls are parallel, there is a free unconstrained direction t_{free} along corridor if there is no other plane constraints. We solve this problem by incorporating with point based SLAM, specifically LSD SLAM [25], to provide photometric odometry constraints along the free direction. We propose the two following combinations.

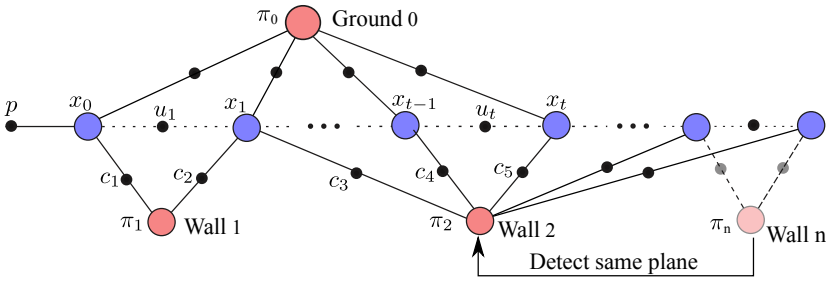


Figure 5.6: Plane SLAM loop closure. After detecting a loop closure, we shift all the factors of plane π_n to π_2 and remove π_n from factor graph.

5.4.1 Depth Enhanced LSD SLAM

This section shows that scene layout understanding could boost the performance of traditional SLAM. LSD SLAM has three main threads: camera tracking, depth estimation and global optimization in Figure 5.7. The core part is depth estimation, determining the quality of other modules. In LSD SLAM, when a new depth map of a keyframe is created, it propagates some pixels' depth from the previous keyframe if it is available. Then the depth map is continuously updated by new frames using multiple-view stereo (MVS). Since our single image pop-up model in Section 5.2 provides each pixel's depth estimation, we integrate its depth into LSD depth map in the following way:

(1) If a pixel has no propagated depth or the variance of the LSD SLAM depth exceeds a threshold, we directly use pop-up model depth.

(2) Otherwise, if a pixel has a propagated depth d_l with variance σ_l^2 from LSD, we fuse it with the pop-up depth d_p of variance σ_p^2 using the filtering approach [118]:

$$\mathcal{N}\left(\frac{\sigma_l^2 d_p + \sigma_p^2 d_l}{\sigma_l^2 + \sigma_p^2}, \frac{\sigma_l^2 \sigma_p^2}{\sigma_l^2 + \sigma_p^2}\right) \tag{5.10}$$

σ_p^2 could be computed by error propagation rule during the pop-up process. In Section 5.2.3, the pixel uncertainty of $\mathbf{u}$ can be modeled as bi-dimensional standard Gaussians Σ_u. If the Jacobian of $\mathbf{p}_c$ *wrt.* $\mathbf{u}$ is J_u from Equation (5.7), then the 3D point's covariance $\Sigma_{p_c} = J_u \Sigma_u J_u^\top$. We find that depth uncertainty σ_p^2 is proportional to the depth square, namely $\sigma_p^2 \propto d_p^2$.

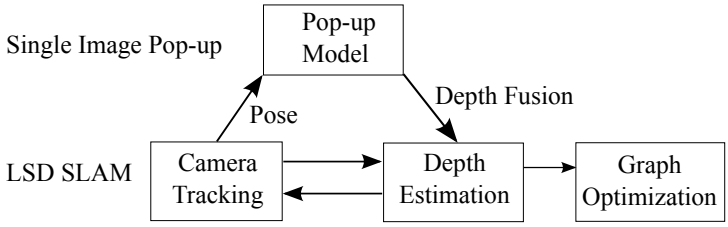


Figure 5.7: *Depth Enhanced LSD SLAM* algorithm that integrates depth estimates from the pop-up model.

Depth fusion could greatly increase the depth estimation quality of LSD SLAM especially at

the initial frame where LSD SLAM just randomly initializes the depth and at the low parallax scenes where MVS depth triangulation has low quality. This is also demonstrated in the latter experiments.

5.4.2 LSD Pop-up SLAM

There has been some work jointly using point and plane as landmarks in one SLAM framework [116] using RGB-D sensors. Currently, we propose a simple version of it to run two stages of SLAM methods. The first stage is *Depth Enhanced LSD SLAM* in Section 5.4.1. We then use its pose output as odometry constraints to run a plane SLAM in Section 5.3. The frame-to-frame odometry tracking based on photometric error minimization could provide constraints along the unconstrained direction in plane SLAM and can also capture the detailed fine movements, demonstrated in the latter experiments.

Figure 5.8 shows the relationship of the three SLAM methods in this paper. The blue dashed box is the improved LSD SLAM: *Depth Enhanced LSD SLAM*. The green and red box show two kinds of plane SLAM. The difference is that *LSD Pop-up SLAM* in this section has additional odometry measurement while *Pop-up Plane SLAM* does not have and usually uses a constant velocity assumption.

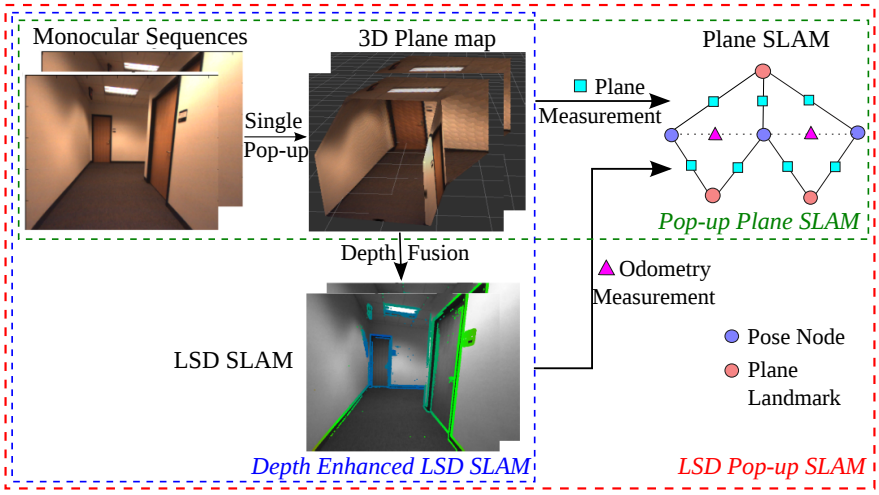


Figure 5.8: Three proposed SLAM methods: (1) *Pop-up Plane SLAM* uses plane measurements from the single image pop-up model. (2) *Depth Enhanced LSD SLAM* is LSD SLAM with depth fusion from the pop-up model. (3) *LSD Pop-up SLAM* is plane SLAM with additional odometry measurements from *Depth Enhanced LSD SLAM*.

5.5 Experiments

We test our SLAM approaches on both the public TUM dataset [119] and two collected corridor datasets to evaluate the accuracy and computational cost. Results can also be found in the

supplemental videos. We compare the state estimation and 3D reconstruction quality with two state-of-art point based monocular SLAM approaches: LSD SLAM [25] and ORB SLAM [1].

5.5.1 TUM SLAM dataset

We choose the TUM *fr3/structure_notexture_far* dataset in Figure 5.1, which is a challenging environment composed of five white walls and a ground plane. We only use RGB images for experiments and use the depth images for evaluation.

Qualitative Comparison

Unfortunately, neither LSD nor ORB SLAM work in this environments because there are only few features and high gradient pixels.

For the *Pop-up Plane SLAM* in Section 5.3, we use the ground truth pose for initialization and a constant velocity motion assumption as odometry measurements. Since the initial truth height is provided, the pop-up model has an absolute scale. Therefore, we can directly compare the pose and map estimates with ground truth without any scaling. The constructed 3D map is shown in Figure 5.1.

Quantitative Comparison

The absolute trajectory estimate is shown in Figure 5.9. This dataset has a total length of 4.58m and our mean positioning error is 0.18 ± 0.07m, with endpoint error 0.10m. From Figure 5.9, our algorithm captures the overall movement but not the small jerk movement in the middle. This is mainly due to the fact that there are only few plane landmarks in SLAM. In addition, *Pop-up Plane SLAM* does not have frame-to-frame odometry tracking to capture the detailed movement, which is commonly used in point based SLAM. In the latter experiments, we show that after getting odometry measurements, state estimation of *LSD Pop-up SLAM* improves greatly.

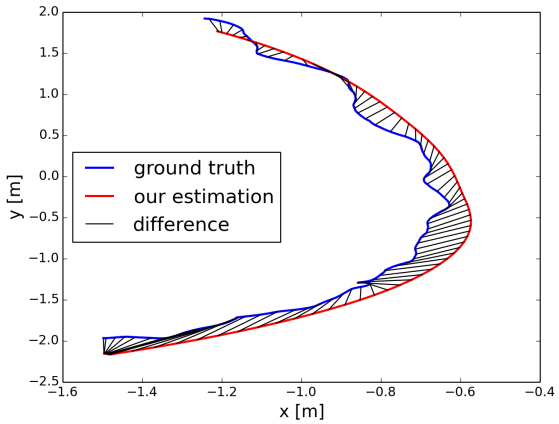


Figure 5.9: Absolute trajectory estimation using *Pop-up Plane SLAM* on TUM *fr3/str_notex_far* dataset. The positioning error is 3.9%, while LSD SLAM and ORB SLAM both fail.

To evaluate mapping quality, we use provided depth maps to compute the ground truth plane position by point cloud plane segmentation using the PCL RANSAC algorithm. The plane normal error is only 2.8° as shown in Table 5.1. We then re-project the 3D plane model onto images to get each pixel's depth estimates. The evaluation result is shown in Figure 5.10 and Table 5.1. The mean pixel depth error is 6.2 cm and 86.8% of the pixels' depth error is within 0.1m.

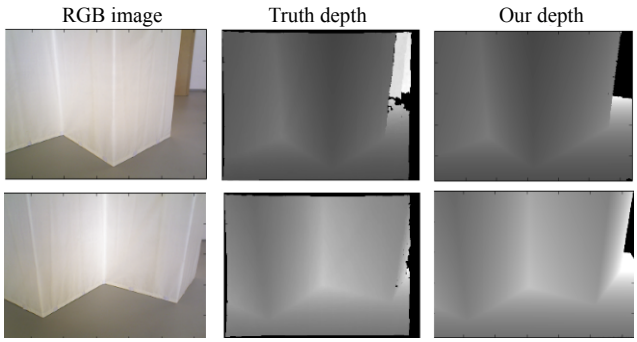


Figure 5.10: Depth reconstruction comparison on TUM dataset.

Table 5.1: 3D Reconstruction Evaluation on TUM dataset.

	Plane normal error	Depth error	Depth error < 0.1m
Value	2.83°	6.2 cm	86.8%

5.5.2 Large Indoor Environment

In this section, we present experimental results using a hand-held monocular camera with a resolution of 640×480 in two large low-texture corridor environments. The camera has a large field of view ($\sim 90^\circ$) which LSD and ORB SLAM typically prefer. Since we do not have ground truth depth or pose, we only evaluate the loop closure error and qualitative map reconstruction. The pose initialization uses the single image rotation estimation in Section 5.2.3 with an assumed height of 1m.

Corridor dataset I

The first dataset is shown in Figure 5.11. LSD SLAM, top center, does not perform well. The best result for ORB SLAM is shown in the top right. Through the tests, we find that even using the same set of parameters, ORB SLAM often cannot initialize the map and fails to track cameras. The randomness results from the RANSAC map initialization of ORB SLAM [1].

Since actual long corridors are easily under-constrained, *Pop-up Plane SLAM* with no actual odometry measurement does not work well. We only provide results for the two other SLAM methods introduced in Section 5.4. *Depth Enhanced LSD SLAM* generates a much better map as shown in the bottom left of Figure 5.11 compared to the original LSD SLAM. Though it is a

semi-dense map, we can clearly see the passageway and turning. Based on that, the *LSD Pop-up SLAM* generates a dense 3D model with distinct doors and pillars.

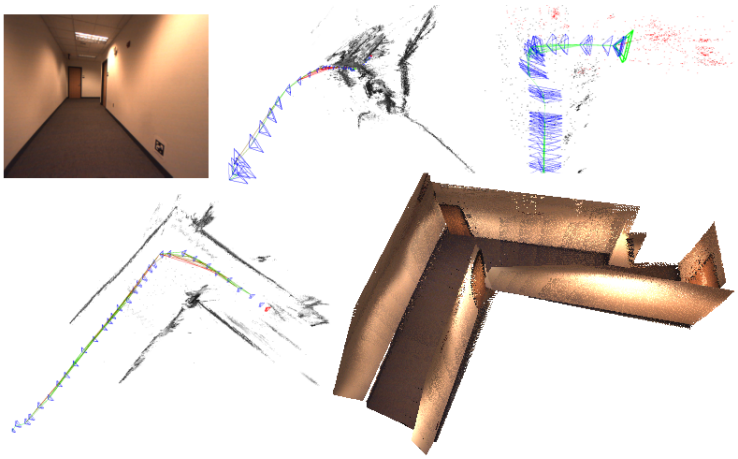


Figure 5.11: Comparison on Corridor dataset I. (top) From left to right: sample frame, LSD SLAM result, ORB SLAM result. (bottom) Our *Enhanced LSD SLAM* in Section 5.4.1, *LSD Pop-up SLAM* result in Section 5.4.2.

Corridor dataset II

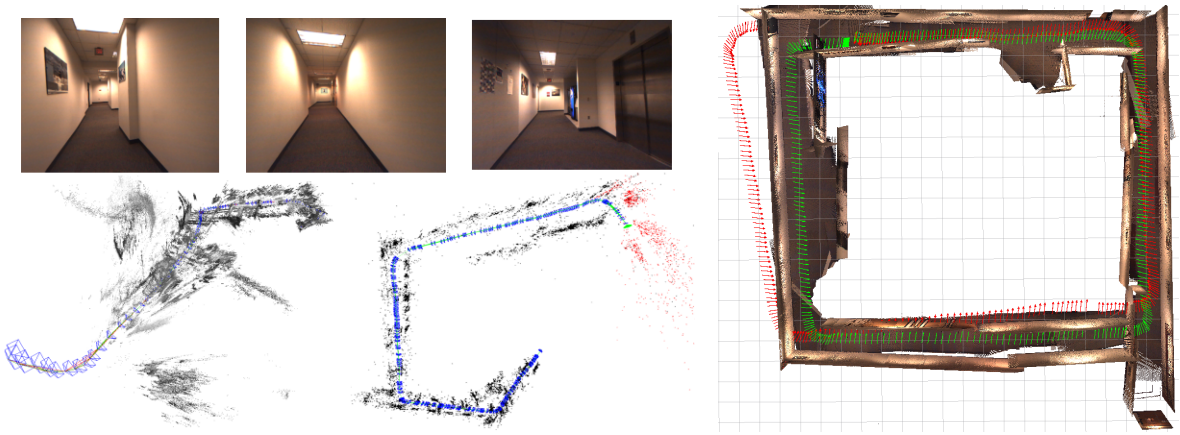
The second dataset is a 60m square corridor containing a large loop shown in Figure 5.12(a). ORB SLAM generates a better map than LSD SLAM, but it does not start tracking until it comes to a large open space with more features and enough parallax.

The result of our algorithms is shown in Figure 5.12(b) where the red line is *Enhanced LSD SLAM* and green line is *LSD Pop-up SLAM*. With the automatic loop closure detection, the *LSD Pop-up SLAM* generates the best 3D map and smallest loop closure error. The grid dimension is $1 \times 1m^2$ in Figure 5.12(b) and the loop closure positioning error is 0.4m of the total 60m trajectory.

5.5.3 Runtime Analysis

Finally, we provide the computation analysis of the Corridor dataset II in Table 5.2. All timings are measured on CPU (Intel i7, 4.0 GHz) and GPU (only for CNN). Most of the code is implemented in C++. Currently, the CNN segmentation, edge detection, and selection consumes 30ms. Note that compared to CNN model in [110], we change the fully connected layers from 4096 to 2048 to reduce the segmentation time by half without affecting the accuracy too much. The iSAM incremental update takes 17.43ms while batch optimization takes 45.35ms. Therefore we only use batch optimization to re-factor the information matrix when a loop closure is detected. In all, our plane SLAM algorithm can run in real time at over 20Hz using single-threaded programming.

We also note that unlike point landmarks, a plane landmark can be observed in many adjacent frames, therefore we actually do not need to pop up planes for each frame. Thus in all the above



(a) (b)

Figure 5.12: Comparison on Corridor dataset II with loop closure. (a)(top) Sample frames in the dataset. (bottom): Other SLAM result: LSD SLAM, ORB SLAM. (b) Our result. Red line: *Enhanced LSD SLAM* result in Section 5.4.1, green line: *LSD Pop-up SLAM* result in Section 5.4.2. Loop closure happens around the top left corner. The grid dimension is 1m. Loop closure positioning error is 0.67%.

pop-up experiments, we process the images at 3Hz (every 10 images), which we find is enough to capture all the planes. This is similar to the keyframe techniques used in many point-based SLAM algorithms.

Table 5.2: SLAM Statistics and Time Analysis on Corridor Dataset II.

Number of planes	146
Number of poses	344
Number of factors	1974
CNN segmentation (ms)	17.8
Edge detection and selection (ms)	13.2
Data association (ms)	<1
iSAM optimization (ms)	17.4
Total frame time (ms)	**49.4**

5.5.4 Discussion

Height effect on map scale

Unlike RGB-D plane SLAM whose plane measurements are generated by the actual depth sensor, our plane measurement and scale is determined by the camera height in the pop-up process in Section 5.2.3. Camera height cannot be constrained by the plane measurements any more and therefore can only be constrained by other information such as odometry measurements or other sensors such as IMU. If other information is inaccurate or inaccessible, the map scale and camera height might drift using the plane SLAM alone. During all the experiments, we did not encounter the scale drift problem because the cameras are kept at a nearly constant height. In the future, we would like to integrate with other sensors.

Ground effect on graph complexity

Since the pop-up process in Section 5.2.3 requires the ground plane to be visible, the ground plane landmark is connected to all the camera poses as shown in Figure 5.4. This will reduce the sparsity of the information matrix and increase the computational complexity in theory. However, it can be alleviated by *variable reordering* before matrix decomposition, for example using the COLAMD algorithm that will force this variable towards the last block column, thereby reducing its impact (see [16]). From the experiments, the ground plane usually increases *fill-in* (added non-zero entries) by only about 10%.

5.6 Conclusions

In this paper, we propose *Pop-up Plane SLAM*, a real-time monocular plane SLAM system combined with scene layout understanding. It is especially suitable for low-texture environments because it can generate a rough 3D model even from a single image. We first improve the previous work to pop up 3D plane model by detecting the true ground-wall edges. Then, we formulate a plane SLAM to optimize a consistent plane map and state estimates across multiple frames. The plane landmark measurement comes from the each image's pop-up model. We utilize the minimal plane representation for optimization and also implement plane SLAM loop closing.

Since plane SLAM itself is easily under-constrained, we propose to integrate it with point based LSD SLAM in two ways: the first is *Depth Enhanced LSD SLAM* by integrating pop-up pixel depth into LSD depth estimation, the second is *LSD Pop-up SLAM*, which is a plane SLAM with odometry measurements from *Depth Enhanced LSD SLAM*

On the public TUM dataset, *Pop-up Plane SLAM* generates a dense 3D map with depth error of 6.2 cm and state estimates error of 3.9% while the state-of-art LSD or ORB SLAM both fail. One another collected 60m long dataset, our loop closure error is only 0.67%, greatly outperforming LSD and ORB SLAM. In addition, it could run in (near) real-time over 10Hz.

In the future, we want to optimize point, edge and plane landmarks in a unified SLAM framework. Besides, more work needs to be done in clutter corridors where ground-wall boundaries may be occluded by objects.

Chapter 6

Cube SLAM: Monocular Object and Plane SLAM

In this chapter, we propose a SLAM system at the level of objects and planes based on the single image semantic scene understanding in Chapter 4. Camera poses are jointly optimized with objects, planes and points in unified bundle adjustment framework. Experiments on various datasets show that SLAM and scene understanding can benefit each other.

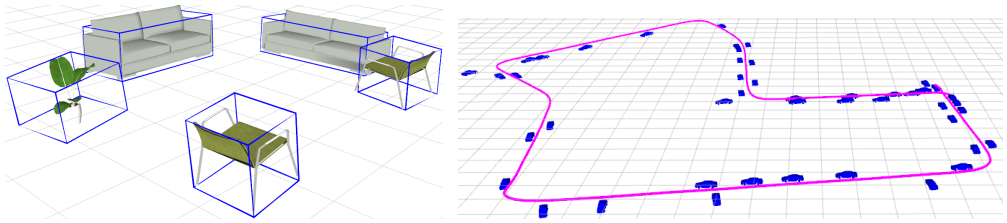
This chapter's work also extends the plane SLAM in Chapter 5 to work in more general environments such as rooms with objects instead of only the corridors.

6.1 Introduction

Simultaneous Localization and Mapping (SLAM) or Structure from Motion (SfM) has been widely used in autonomous robot navigation and Augmented Reality (AR) applications. The classic SLAM approach is to track visual geometric features such as points [1], lines [30], planes [32] across frames then minimize the reprojection or photometric error through bundle adjustment (BA). ORB SLAM [1] and DSO [2] are two representative state-of-the-art SLAM algorithms. Most existing SLAM approaches represent the environments as a point cloud, either sparse or semi-dense, which may not satisfy many high level and intelligent tasks. For example in autonomous driving, vehicles need to be detected in 3D space to keep safety and in AR application, 3D objects and layout planes also need to be localized for more realistic physical interactions.

There are typically two categories of approaches to combine visual understanding and SLAM. The decoupled approach first builds the SLAM point cloud then further labels [63] [120] or detects 3D objects [7] and planes [121], while the coupled approach jointly optimizes the camera pose with the object and plane location. Most existing object SLAM such as SLAM++ [5] [8] requires prior object models to detect and model the object, which limits the application in general environments. Some prior works also utilize architectural planes for dense 3D reconstruction but mostly rely on RGBD [10] sensors or LiDAR scanner [122].

In additiona, most existing SLAM approaches assume the environment to be static or mostly static. Features from dynamic regions are often treated as outliers and not used for camera



(a) (b)

Figure 6.1: Monocular 3D object detection and mapping without prior object models. Mesh model is just for visualization and not used for detection. (a) ICL NUIM data with various objects, whose position, orientation and dimension are optimized by SLAM. (b) KITTI 07. With object constraints, our monocular SLAM can build a consistent map and correct scale drift, without loop closure and constant camera height assumption.

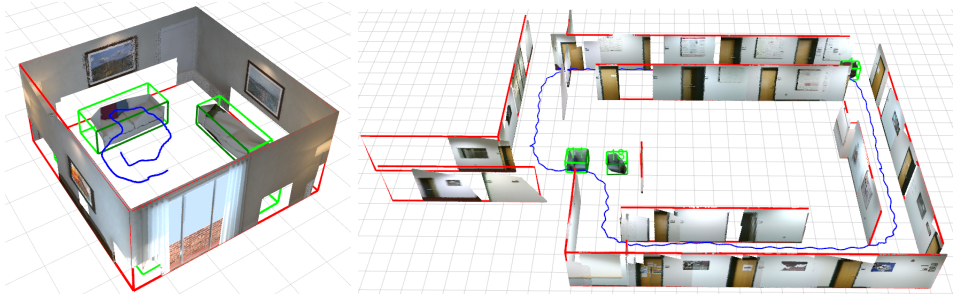


Figure 6.2: Example result of dense SLAM map with points, objects (green box), planes (red rectangle) reconstructed using only a monocular camera. (top) ICL living room dataset. (bottom) Collected long corridor dataset.

pose estimation [1], however this assumption may not hold in many practical environments. For example, there are many other moving vehicles and pedestrians on the road. It is also an important task to detect and predict the trajectory of moving objects in many applications.

In this chapter, we develop a new object and plane SLAM without prior object shape priors in both static and dynamic environments. More importantly, we demonstrate that these high level landmarks can improve both camera pose estimation and dense mapping. From the single image understanding in Chapter 4, we can detect the cuboid object and layout planes from a single image. They are treated as SLAM landmarks and optimized together with points and cameras in multi-view BA. Objects and planes are utilized in two fold: firstly, provide geometry, scale and semantic constraints in BA for example Manhattan world assumption and object supporting relationships; secondly, provide depth initialization for points difficult to triangulate. The estimated camera poses from SLAM also benefit the single-view object and plane detection. Lastly in the dynamic case, instead of treating moving objects as outliers, we jointly optimize the trajectories of camera and objects based on dynamic point observation and motion model constraints. In summary, our contributions are as follows:

- An novel object SLAM method with novel measurements between cameras, objects and points, achieving better pose estimation on many datasets including KITTI benchmark.

- An algorithm to utilize moving objects to improve pose estimation in dynamic scenarios.
- The first monocular SLAM method incorporating points, objects and planes, and show improvements on both localization and mapping over state-of-the-art algorithms. New plane measurement and boundary updates are also proposed.

In the following, we first introduce the related work followed by method overview in Sec 6.2. We then explain three main parts: static object SLAM in Sec 6.3, dynamic object SLAM in Sec 6.4, object and plane SLAM in Sec 6.5. Implementation details and experiments are presented in Sec 6.6. Conclusion is made in Sec 6.8.

Closely Related Work

Object and Plane SLAM We here briefly review the object and plane SLAM. A more complete review can be found in Chapter 2.

There are typically two categories of them, either decoupled or coupled. The decoupled approaches first build SLAM point cloud map then further detect 3D object and planes based on point cloud clustering and image evidence filtering [65] [65] [7]. It shows improvement compared to 2D object detections but doesn't change SLAM part, thus it may fail if SLAM cannot build a high quality map. The coupled approach is usually called object and plane level SLAM. Salas-Moreno *et al.* proposed a practical SLAM system called SLAM++ using RGB-D cameras and prior object models [5]. Recently, a real time monocular object SLAM using the prior object models was proposed in [8]. QuadriSLAM [12] proposed an online SLAM without prior models. Lee *et al.*[10] estimated the layout plane and point cloud registration iteratively to reduce RGB-D mapping drift. McCormac *et al.* proposed an online volumetric object-level SLAM Fusion++, without prior shape models using RGBD camera [11]. Recently, [13] proposed a similar work to jointly optimize objects, planes, points with camera poses. The difference is that we use monocular camera instead of RGBD camera and also have different object representations.

Dynamic environment SLAM SLAM in the dynamic environment has been a challenging problem. Most existing approaches treat dynamic region features as outliers and only utilize static background for pose estimation [1] [123] [124]. After static SLAM is solved, some other works additionally detect, track, and optimize the trajectory of dynamic objects in order to build a complete 3D map [51] [125] [126]. The optimization is based on the object's reprojection error, object motion model and so on. However, these approaches are likely to fail in highly dynamic environments due to the lack of reliable static background features.

There is also one recent work utilizing dynamic point BA to improve camera pose estimation, based on the rigid shape and constant motion assumption [127], however, the paper showed limited real dataset results and didn't have object representation in the map.

6.2 Method Overview

We extend the single image object and plane understanding to multi-view object and plane SLAM. The system is built on feature point based ORB SLAM2 [1], which includes the front-

end of camera tracking and back-end of BA. Our main change is the modified BA to include objects, planes, points and camera poses together, which will be explained in more details in this section. Other SLAM implementation detail is in Section 6.6.

6.2.1 BA Formulation

As we know, BA is the process to jointly optimize different map components such as camera poses and points [1] [2]. Consider a set of camera poses $C = \{c_i\}$, 3D landmark objects $O = \{o_j\}$ and planes $\Pi = \{\boldsymbol{\pi}_i\}$. Points $P = \{p_k\}$ are also used in most of our experiments because objects alone usually cannot fully constrain camera poses. Then BA can be formulated as nonlinear least squares optimization problem:

$$C^*, O^*, \Pi*, P^* = \arg \min_{\{C,O,\Pi,P\}} \sum_{i \in C, j \in O, k \in \Pi, m \in P} \mathbf{e^T} W \mathbf{e} \tag{6.1}$$

where $\mathbf{e}$ is the measurement error between each element. W is the weight matrix, or called information matrix, of different error functions. Definitions of variables and errors are in the following. Then the optimization problem can be solved by many existing libraries such as g2o and iSAM.

6.2.2 Parameterization

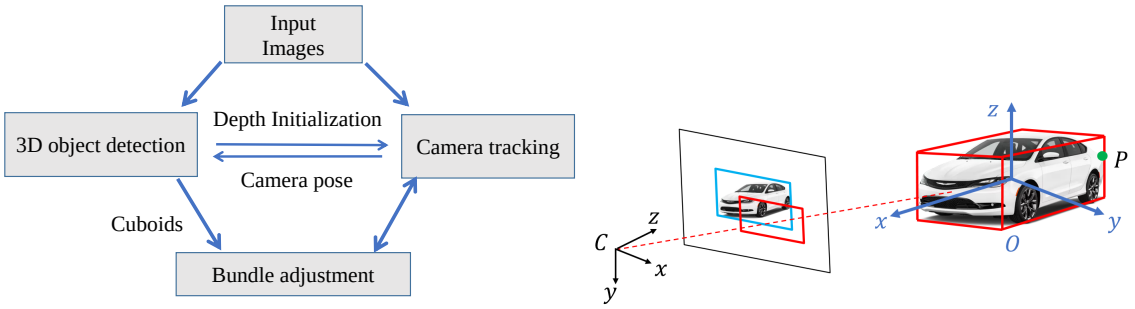
There are four different components in the map: camera, point, object and plane.

Cameras and Points The standard camera poses are represented by $T_c \in SE(3)$ and points are represented by $P \in \mathbb{R}^3$.

Objects As explained in the cuboid detection Section 4.2.1 in Chapter 4, objects are modelled as 9 DoF parameters: $O = \{T_o, D\}$ where $T_o \in SE(3)$ is 6 DoF pose, and $D \in \mathbb{R}^3$ is the cuboid dimension. Other shapes can also be used for example ellipsoids [12]. In some environments such as KITTI, we can also use the provided object dimension then D is not needed to optimize. Subscript m indicates measurement. The coordinate system is shown in Fig 6.3(b).

Planes We adopt the infinite plane in [32] which represents plane as a quaternion $\boldsymbol{\pi} = (\mathbf{n}^\top, d)^\top$ st. $\|\boldsymbol{\pi}\| = 1$, suitable for graph optimization. n is the plane normal and d is plane distance to the origin. In some environments, we can use the Manhattan assumptions, namely the plane normal is fixed and parallel to one of the world frame axes, therefore only one number d is needed to represent it in that case.

Dynamic scenarios For the dynamic object O^i, we need to estimate its pose ${}^jO^i$ in each observed frame j. For dynamic point P^k, we represent its position anchored on the object O^i as ${}^iP^k$, which is fixed based on the rigidity assumption. Its world pose will change over time and is not suitable for SLAM optimization.



(a) (b)

Figure 6.3: (a) Our object SLAM pipeline. Single view object detection provides cuboid landmark and depth initialization for SLAM while SLAM can estimate camera pose for more accurate object detection. (b) Coordinate system. Measurement errors between cameras, objects and points during BA.

6.3 Static Object SLAM

In this section, we mainly deal with static objects and the dynamic scenarios are addressed in the next section.

6.3.1 Measurements

Object-camera measurement

We propose two kinds of measurement errors between objects and cameras.

The first is 3D measurement utilized when the 3D object detection is accurate for example if RGBD sensor is used. The detected object pose $O_m = (T_{om}, D_m)$ from single image detection in Section 4.2.1 serves as the object measurement from the camera frame. To compute its measurement error, we transform the landmark object to the camera frame then compare with the measurement shown as:

$$e_{3D} = \| \log \left((T_c^{-1} T_o) T_{om}^{-1} \right)^{\vee}_{\mathfrak{se}_3} \| + \| D - D_m \|_2 \tag{6.2}$$

where $\log$ maps the $SE3$ pose error into tangent vector space suitable for optimization. The two parts represents the pose error and dimension error and are all in meter unit, except the rotation part. Therefore, we give them same weight. Huber robust cost function is applied to all measurement error to improve the robustness [1].

We need to note that, without prior object model, our image-based cuboid detection cannot differentiate between the front or back of objects. For example, we can represent the same cuboid by rotating the object coordinate frame by $90°$ and swapping length with width value. Therefore, we need to rotate along the height direction for $\pm 90°, 0, 180°$ to find the smallest error in Eq 6.2.

The second is 2D measurement. We can project the cuboid landmark onto the image plane to get the 2D bounding box shown as the red rectangles in Fig. 6.3(b) then compare it with the

blue detected 2D bounding box:

$$e_{2D} = \| \ (c, d) - (c_m, d_m) \ \|_2 \tag{6.3}$$

where (c, d) is the center and dimension of the 2D box. This measurement error is in image pixel unit and has much less uncertainty compared to the 3D error in Eq 6.2 because 2D object detection is usually more accurate compared to 3D detection. This is similar to projecting map points onto images to formulate reprojection error. But it also loses information after projection because many different 3D cuboids can project to the same 2D rectangle thus more observations are needed to fully constrain the camera poses and cuboids.

Modelling and estimating the error weight matrix W is not straightforward compared to points due to the complicated detection procedure. Therefore we simply give more weights to the semantic confident and geometric close objects. Suppose the cuboid-camera distance is d and the object's 2D detection probability is p, then we can define $w = p \times \min((70 - d), 0)/50$ on KITTI data, where 70 is truncation distance. Parameters may vary with different datasets.

Object-point measurement

Points and objects can provide constraints for each other. If point P belongs to an object shown in Fig. 6.3(b), it should lie inside the 3D cuboid. Thus we can first transform the point to the cuboid frame then compare with cuboid dimensions:

$$e_{op} = \max(|(T_o^{-1} P)| - D_m, 0) \tag{6.4}$$

where $\max$ operator is used because we only encourage points to lie inside cuboid instead of exactly on surfaces.

Point-camera measurement

We use the standard 3D point re-projection error in feature-based SLAM [1].

6.3.2 Data association

Data association across frames is another important part for SLAM. Compared to point matching, object association seems to be easier as more texture is contained and many 2D object tracking or template matching approaches can be used. Even simple 2D box overlapping can work in some simple scenarios. However, these approaches are not robust if there is severe object occlusion with repeated objects as shown in Fig 6.4. In addition, dynamic objects need to be detected and removed from current SLAM optimization but standard object tracking approaches cannot classify whether it is static or not, unless specific motion segmentation is used.

We thus propose another method for object association based on point matching. For many point based SLAM [1], dynamic points can be effectively detected through descriptor matching and epipolar line checking. Thus we first associate points to their belonged objects if points are observed enough times of being in the 2D object bounding box and close to cuboid centroid in 3D space. Some latest instance segmentation [35] could also be used to speed up and improve the

Figure 6.4: Object association in dynamic and occluded scenarios in KITTI 07. Green points are normal map points, and other color points are associated to objects with the same color. The front cyan moving car is not added as SLAM landmark as no feature point is associated with it. Points in object overlapping areas are not associated with any object due to ambiguity.

point-object association accuracy. Then we can find object matching which has the most number of shareable map points and also exceed a threshold (10 in our implementation). Note that this object-point association is also used when computing object-point measurement error during BA in Sec 6.3.1. Through our experiments, this approach works well for wide baseline matching, repetitive objects, occlusions, and dynamic scenarios explained in Fig 6.4.

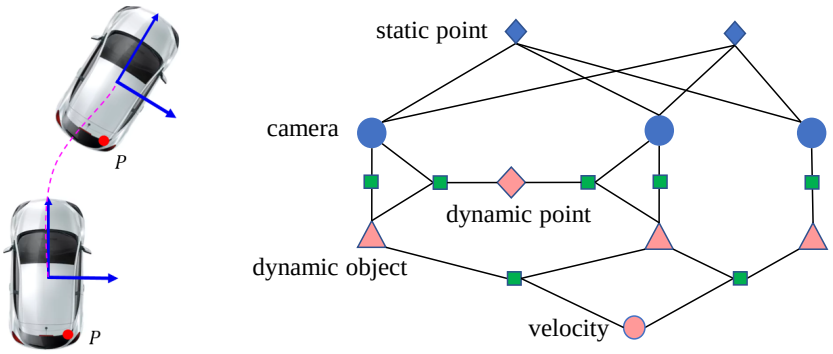
6.4 Dynamic Object SLAM

The previous section deals with the static object SLAM and in this section, we propose an approach to jointly estimate the camera pose and dynamic objects' trajectories. We need to make some prior assumptions about the objects otherwise there are too many degrees of freedom that need to be resolved.

As shown in Fig 6.5(a), there are two commonly used assumptions that the object is rigid and follows some physically feasible motion model. The rigid body assumption indicates that a dynamic point's local position on its belonged object doesn't change over time. This allows us to utilize the standard 3D map point reprojection error to optimize its position. For the motion model, the simplest form is constant motion model with uniform velocity. For some specific object such as vehicles, it is additionally constrained to follow the nonholonomic wheel model (with some side-slip).

6.4.1 Measurements

The factor graph of the dynamic object estimation is shown in Fig .6.5(b). Blue nodes are the static SLAM components while the red ones represent the dynamic objects' poses in each frame, dynamic points' positions in local object frame and motion velocity, including velocity and steering angle. The green squares are the measurement factors including dynamic point observation and object motion models explained as follows. With these factors, camera poses can also be constrained by the dynamic elements.



(a) (b)

Figure 6.5: Dynamic object SLAM factor graph. Blue nodes represent the static SLAM component and red ones represent new dynamic variables. The green squares are the factors of dynamic elements.

Object motion model

The general 3D motion can be represented by a pose transformation matrix $T \in SE(3)$. We can apply T to the previous pose then compute pose error with the current pose. We here explain a more restricted car motion model [128] used in our experiments. Car motion can be represented by linear velocity v and steering angle ϕ. Suppose the vehicle runs on a local planar surface approximately, then only x, y, θ (heading) is needed to represent its state. Then the predicted state is:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + v\Delta t \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ \tan(\phi)/L \end{bmatrix} \tag{6.5}$$

where L is the distance between the front and rear wheel center. Note that this model requires that x, y, θ is defined at the rear wheel center while our object frame is defined at the vehicle center. The two coordinate frames have $L/2$ offset that needs to be compensated. The final error is simply as:

$$e_{mo} = [x', y', \theta'] - [x, y, \theta]; \tag{6.6}$$

Dynamic point observation

As explained before, the dynamic point is anchored to its belonged object, thus it is first transformed to the world frame then projected onto the camera. The reprojection error for the kth point on ith object wrt. the jth image is:

$$e_{dp} = \pi({}^{j}T_o^i *^i P^k, T_c^j) - z_{kj} \tag{6.7}$$

which ${}^{j}T_o^i$, ${}^{i}P^k$, T_c^j represent the object, point and camera pose respectively. z_{kj} is the measurement pixel and π is the camera projection function.

6.4.2 Data association

Through the experiments, we find that the association method for static environments in Section 6.3.2 is not suitable for the dynamic cases due to the difficulty in matching dynamic point features. The typical way to track a feature point is to predict its projected position, search nearby features match descriptors then check epipolar geometry constraints [1]. However, for monocular dynamic cases, it is difficult to predict the accurate movement of objects and points and the epipolar geometry is also not accurate when object motion is inaccurate.

We thus design different approaches for the point and object association. The feature points are directly tracked by KLT sparse optical flow algorithm, which doesn't require the 3D point position. After pixel tracking, the 3D position of the features will be triangulated considering the object movement. However, KLT tracking might fail when the pixel displacement is large, for example when another vehicle comes close and towards the camera. Therefore for the dynamic object tracking, we don't utilize shared feature point matching approach, instead, we directly utilize visual object tracking algorithm [129]. The object's 2D bounding box is tracked and its position is predicted from the previous frame, then it is matched to detected bounding box in the current frame with the largest overlapping ratio.

6.5 Object and Plane SLAM

This section is an extension to object SLAM in Sec 6.3 to include planes as SLAM landmarks. Similar as before, the new measurement functions and association are first explained.

6.5.1 Measurements

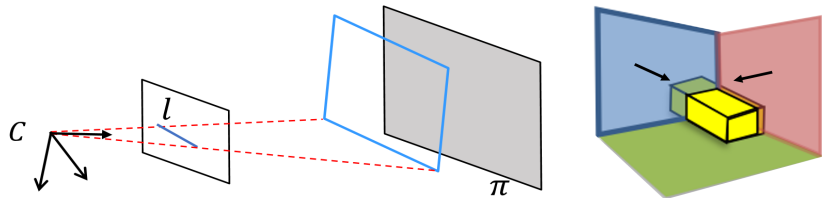
Different constraint functions between the map elements are proposed to formulate a factor graph optimization. Camera-point observation model is standard reprojection error [1].

Camera-plane

Different from RGBD based plane SLAM which can directly get plane measurements from point cloud plane fitting [13] [32], we need to pop up the plane to get local plane measurement which depends on the camera pose. [33] updates the measurement after graph optimization which is not an optimal solution. Therefore we update it in each iteration during the optimization. Denote the wall-ground edge as l, then plane error is defined as:

$$e_{cp} = \| \log \left(\boldsymbol{\pi}_{obs}(l, T_c), \boldsymbol{\pi} \right) \| \tag{6.8}$$

Where $\boldsymbol{\pi}_{obs}$ is the process of projecting ground edge l onto 3D ground shown as blue plane in Fig .6.6(a). It also depends on camera pose T_c. Then the generated plane is compared with plane landmarks $\boldsymbol{\pi}$ using $\log$ quaternion error defined in [32].



(a) (b)

Figure 6.6: Object-plane SLAM observation functions. (a) Camera-plane observations. The detected ground edge is projected to 3D space to compare with landmark plane. (b) Plane-object observation error depends on the object volume occluded by planes. Camera-object observation is in Fig 6.3(b).

Camera-object

We follow the cuboid observation functions defined in the prior work [109] (Sec 6.3). The cuboid landmark is projected onto the image plane then compared with the detected 2D bounding box shown Fig 6.3(b).

$$e_{2D} = \| \ (c, d) - (c_m, d_m) \ \|_2 \tag{6.9}$$

where (c, d) is the center and dimension of the 2D box. This 2D measurement error has much less uncertainty compared to 3D cuboid error as explained in [109]. To make the optimization robust, different weights are assigned to different objects' error. More weight is given to semantic confident and geometric close objects.

Object-plane

There are different forms of object-plane constraints depending on the environment assumptions for example objects are supported by planes [13] or object orientation matches the nearby plane normal. We here design a weaker but more general constraint that objects should not be occluded by nearby planes in the camera view shown in Fig 6.6(b). If the plane normal is defined to point to the camera, then the object-plane occlusion error is defined as:

$$e_{op} = \sum_{i=1:8} \max(0, -\boldsymbol{\pi} P_{oi}) \tag{6.10}$$

Where P_{oi} is one of the eight cuboid corners. If the cuboid lies on the front side of plane (towards camera), $e_{op} = 0$.

Point-plane

It is usually difficult to accurately detect if points belong to a plane from 2D images as layout planes are usually background and points may belong to foreground objects. To improve the robustness, we first select points in the 2D wall plane polygon then filter out points that are farther away from the 3D plane than a threshold. The point-plane error is defined as:

$$e_{pp} = \|\boldsymbol{\pi} P\| \tag{6.11}$$

Note that to be robust to outliers, huber loss is applied to all above error functions.

6.5.2 Data association

Data association for different landmarks across multiple views is important for SLAM. For point association, we use the point feature matching in ORB SLAM [1]. Object association follows the work in the previous Section 6.3.2. Each object contains a set of feature points belonging to it then we can find object matching which has the most number of shared map points exceeding a threshold (10 in our implementation). This approach is easy to implement and can also easily detect dynamic objects.

Plane association is based on the two following criteria. One is the geometry information such as the plane normal angle difference and plane distance to each other. The other is the shared feature points matching similar to objects. The point plane belonging relation is also used when computing point-plane error in Eq 6.11.

6.6 Implementation

Static Object SLAM The pipeline of the whole SLAM algorithm is shown in Fig. 6.3(a). As mentioned in Sec 6.3, our system is based on ORB SLAM2 and we didn't change camera tracking and keyframe creation module. For the newly created keyframe, we detect the cuboid objects, associate them, then perform bundle adjustment with camera poses and points. For the dynamic objects, we can choose to reconstruct or ignore them depending on different tasks. The cuboid is also used to initialize depth for feature points difficult to triangulate when stereo baseline or parallax is smaller than a threshold. This can improve the robustness in some challenging scenarios such as large camera rotations demonstrated in the experiments. Since the number of objects is far less than points, object association and BA optimization runs efficiently in real time. To get an absolute map scale for monocular SLAM, the initial frame's camera height is provided to scale the map. Note that our object SLAM can also work independently without points. In some demonstrated challenging environments with few feature points, ORB SLAM cannot work, but our algorithm can still estimate camera poses using only the object-camera measurement.

Dynamic Object SLAM The implementation of dynamic objects mostly follows the previous section with some difference. The constant motion model assumption may not hold for practical datasets because objects may accelerate and decelerate for example in Fig 6.11. Through the ground truth object velocity analysis, we find that the velocity roughly stays the same in recent 5 seconds. Therefore, in our SLAM, motion model is only applied to recent 5 seconds observations.

Object and Plane SLAM For plane proposals, we first detect and merge line segments then remove lines shorter than 50 pixels and more than 50 pixels away from the wall-ground segmentation boundary.

For the SLAM part, our system is built on the feature point based ORB SLAM, augmented with objects and planes. We compute the jacobians of the newly created observation functions then perform BA using g2o library. The explicit image recognition based loop closure in ORB SLAM is disabled to better show the improvements by objects and planes. Since the number of objects and planes is far less compared to point features, the overall BA optimization is still efficient enough to run in real time. Meanwhile, outlier objects or planes usually have more severe effects compared to outlier points, thus strict outlier rejections need to be used. The object and plane landmark will be deleted if it has not been observed 3 times in recent 15 frames after creation or if there are less than 10 stable feature points associated with it, except the white wall surfaces with few 2D features initially. In most of the experiments, we use the Manhattan plane representation with a fixed surface normal in Section 6.2.2 to improve the performance. If the initial generated wall surface normal difference with Manhattan direction exceeds 30 degrees, it will also be treated as outliers.

In addition to being used as SLAM landmarks, objects and planes can also provide depth initialization for feature points. When the inlier feature point ratio (features matched to the map divided by total features) is below 0.3, we create some new map points directly using depth from objects and planes. This can improve monocular SLAM performance in low texture environments and large rotation scenarios.

Different from the prior monocular plane SLAM [33], ground plane is not used in this work because there is no actual edge measurement corresponding to the ground plane.

Objects and planes can also benefit the final dense mapping. As mentioned before, the plane is represented as infinite plane in the BA optimization. However, for visualization and final dense mapping, we need to keep track of the plane boundary. The initial plane boundary is computed from the back-projected line segments. Since a plane landmark contains some map points which are already associated for BA optimization, we can update plane boundary based on the points' positions so that they all lie in the plane boundary. For the final dense mapping, we also intersect nearby planes to form valid corners shown in Fig 6.2. The intersection pairs are determined based on the image line segments. If endpoints of two line segments are close enough shown in Fig 4.1, and their final 3D plane normal and boundary pass some restrictive tests, they will intersect to form new boundaries. We then back-project the plane region pixels, excluding object areas, onto the optimized plane landmark. For feature points belonging to objects, we create triangular meshes in 3D space to get dense mapping. Note that in the SLAM optimization, planes are represented as infinite planes but for visualization purposes, we need to keep track of the plane boundary polygon.

BA Weight Parameter As shown in Eq 6.1, our SLAM optimization has many new types of measurement errors in addition to the commonly used camera-point reprojection error. Therefore, we need to tune the weight parameters W between different errors. We mainly determine the weights based on the measurement magnitude and the number of measurements of specific types. For example, the single cuboid re-projection error is usually around 50 pixels, much

larger compared to points, but the number of object measurements is also much less compared to points. If camera-point measurement weight is 1, camera-object weight is usually between 0.5 2 and camera-plane weight is around 1-4 after experimental searches.

In addition to the weights of different measurement types, we also need to assign weight to specific objects and planes as some might be quite inaccurate compared to others. We generally assign more weights to nearby objects and planes because the detection is usually more accurate for closer objects. Some other factors can also be considered for example the object 2D detection probability and cuboid fitting error, but based on our experiments, these have small effects on the final result.

6.7 Experiments

6.7.1 Static Object SLAM

We first evaluate the performance of static object SLAM, including camera pose estimation, and 3D object IoU after BA optimization. We show that SLAM and object detection can benefit each other in various datasets. Root mean squared error (RMSE) [119] and KITTI translation error [107] are used to evaluate the camera pose. Note that even though our algorithm is monocular SLAM, we can get the map scale from the first frame's camera height, therefore, we directly evaluate the absolute trajectory error without aligning it in scale. We turn off the loop closure module of ORB SLAM in all the following experiments when using and comparing, to better evaluate the monocular pose drift. ORB SLAM-No LC (loop closure) is used to denote it in all following tables.

TUM RGBD and ICL-NUIM dataset

These datasets [119] [130] provide ground truth camera pose trajectory and only RGB image is used in SLAM. We register a global point cloud using depth image and manually label 3D cuboids as the ground truth object.

We first test on TUM *fr3_cabinet* shown in Fig. 6.7 which is a challenging low texture dataset and existing monocular SLAM algorithms all fail on it due to few point features. Object is the only SLAM landmark and the 3D object-camera measurement in Sec 6.3.1 is used because it can provide more constraints than 2D measurement. The left of Fig. 6.7 shows our online detected cuboid in some frames using estimated camera pose from SLAM. There is clearly large detection error in the bottom image. After multi-view optimization, the red cube in the map almost matches with the ground truth point cloud. From row "fr3/cabinet" in Table. 6.1, 3D object IoU is improved from 0.46 to 0.64 after SLAM optimization compared to the single image cuboid detection. The absolute camera pose error is 0.17m.

We then test on ICL living room dataset which is a general feature rich scenario. Since there is no absolute scale for monocular DSO or ORB SLAM, we compute their pose error after scale alignment [2]. We improve the object detection accuracy while sacrificing a bit camera pose accuracy due to imperfect object measurements. As can be seen from the mapping result of ICL

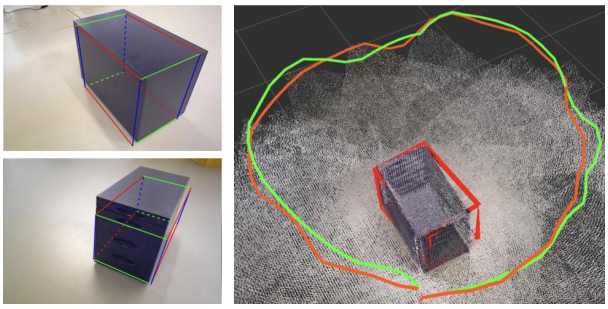


Figure 6.7: Object SLAM results on TUM fr3_cabinet. Red cube is the optimized object landmark, matching well with the truth point cloud. Red and green trajectories are the predicted and truth camera paths. Existing SLAMs fail on this dataset due to low texture.

Table 6.1: Object Detection and SLAM Result on Indoor Datasets

Dataset	Object IoU		Pose error (m)		
	single view	after BA	DSO *	ORB-No LC *	Our
fr3 cabinet	0.46	**0.64**	X	X	**0.17**
ICL room2	0.33	**0.49**	**0.01**	0.02	0.03
Two Chair	0.37	**0.58**	0.01	X	**0.01**
Rot Chair	0.35	**0.50**	X	X	**0.05**

* Pose error with scale alignment.

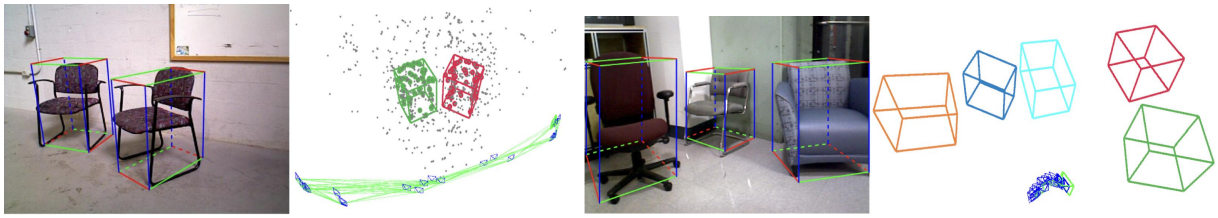
data in Fig. 6.1(a), our approach is able to detect different objects including sofas, chairs, and pot-plant demonstrating the advantage of our 3D detection without prior models.

Collected chair dataset

We collect two chair datasets using Kinect RGBD camera shown in Fig. 6.8. The RGBD ORB SLAM result is used as the ground truth camera poses. The second dataset contains large camera rotation which is challenging for most monocular SLAM. As shown in Fig. 6.8(a), after optimization, cuboids can fit tightly with the associated 3D points showing that object and point estimation benefit each other. The quantitative error is shown in the bottom two rows of Table. 6.1. DSO is able to work in the first dataset but performs very bad in the second one due to large camera rotation. Mono ORB SLAM fails to initialize in both cases while our cuboid detection can provide depth initialization for points even from a single image. Similar as before, the 3D object IoU is also improved after BA.

KITTI Dataset

We test on two kinds of KITTI dataset, one is the short sequence with provided ground truth object annotations, the other is the long sequence of standard odometry benchmark without object annotations. The 2D object-camera measurement in Sec 6.3.1 is used for BA because of its low uncertainty compared to 3D measurements for vehicle detection. We also scale ORB SLAM's



(a) (b)

Figure 6.8: Object SLAM on Collected chair datasets. (a) Objects fit tightly with the associated points. (b) Objects improve camera pose estimation when there is large camera rotation.

Table 6.2: Object Detection and Camera Pose Estimation on KITTI Raw Sequence

Seq		22	23	36	39	61	64	93	95	96	117	Mean
Object	Single view[50]	0.52	0.32	0.50	0.54	**0.54**	0.43	0.43	0.40	0.26	0.25	0.42
	Ours before BA	0.55	**0.36**	0.49	0.56	0.54	0.42	**0.46**	0.49	0.20	0.30	0.44
3D IoU	Ours BA	**0.58**	0.35	**0.54**	**0.59**	0.50	**0.48**	0.45	**0.52**	**0.29**	**0.35**	**0.47**
Trans	ORB -No LC	13.0	**1.17**	7.08	6.76	**1.06**	7.07	4.40	**0.86**	3.96	4.10	4.95
error(%)	Ours	**1.68**	1.72	**2.93**	**1.61**	1.24	**0.93**	**0.60**	1.49	**1.81**	**2.21**	**1.62**

initial map by the first frame camera height (1.7m in our implementations) in order to evaluate its absolute pose error. In Fig 6.9, we can observe that the initial trajectory segment before first turning matches well with ground truth, indicating the initial map scaling is correct. For KITTI dataset, we additionally initialize object dimension using prior car size ($w = 3.9, l = 1.6, h = 1.5$ in our implementation) to maintain long-term scale consistency, which is also used in other object SLAM works [69][70]. This is especially useful when objects are not observed frequently in some sequence.

For the first category data, we select 10 kitti raw sequences with the most number of ground truth object annotations named "2011_0926_00xx". The ground truth camera pose is from the provided GPS/INS poses on KITTI. For the object IoU, we compare three methods. One is the single image cuboid detection from [50]. Second is the object poses just using data association between frames. For example, if an object in one frame is far away, the 3D detection may be inaccurate but in another frame, the same object is closer thus the 3D detection becomes more accurate. Therefore, correct camera pose estimation and data association should also improve the object IoU. Third, the object poses after BA optimization are also evaluated shown in the third row. Most of the time the object accuracy increases after data association and optimization as shown in the top three rows of Table 6.2, however, in some sequences, due to local position drift, the object IoU may also decrease a bit. For camera pose estimation, object SLAM can provide geometry constraints to reduce the scale drift of monocular SLAM.

For the KITTI odometry dataset, most existing monocular SLAMs use constant ground plane height assumption to reduce monocular scale drift [131] [132]. Recently, there are also some object based scale recovery approaches [69] [70]. Results of them are taken from their paper directly. We didn't compare with ORB SLAM in this case, as without loop closure, it cannot

Table 6.3: Camera Pose Estimation Error on KITTI Odometry Benchmark

Seq			0	2	3	4	5	6	7	8	9	10	Mean
Trans Error (%)	Ground based	[131]	4.42	4.77	8.49	6.21	5.44	6.51	6.23	8.23	9.08	9.11	6.86
		[132]	**2.04**	**1.50**	3.37	1.43	2.19	**2.09**	-	**2.37**	**1.76**	**2.12**	2.03
	Object based	[70]	3.09	6.18	3.39	32.9	4.47	12.5	2.81	4.11	11.2	16.8	9.75
		Ours	2.40	4.25	**2.87**	**1.12**	**1.64**	3.20	**1.63**	2.79	3.16	4.34	2.74
	Combined	Ours	1.97	2.48	1.62	1.12	1.64	2.26	1.63	2.05	1.66	1.46	**1.78**
RMSE (m)	Object based	[69]	73.4	55.5	30.6	10.7	50.8	73.1	47.1	72.2	31.2	53.5	49.8
		Ours	13.9	26.2	3.79	1.10	4.75	6.98	2.67	10.7	10.7	8.37	**8.91**

recover scale in the long sequence and has significant drift error shown in Fig 6.9. As shown in Table 6.3, our object SLAM performs much better compared to other SLAM using objects, because they represent vehicles as spheres or only utilize vehicle height information, which is not accurate compared to our cuboid BA. Our algorithm is also comparable to ground-based scaling approaches. Visualization of some object mapping and pose estimation are shown in Fig 6.1(b) and Fig 6.9, where we can see our approach greatly reduces monocular scale drift.

It performs worse in some sequences such as Seq 02, 06, 10, mainly because there are not many objects visible over long distance thus causing scale drift. Therefore, we also propose a simple method to combine ground height assumption with our object SLAM. If there are no objects visible in recent 20 frames, we do point cloud ground plane fitting then scale camera poses and local map based the constant ground height. As shown in row "Combined" in Table 6.3, it achieves the state-of-art accuracy of monocular SLAM on KITTI benchmark. Note that ground plane based approaches also have their limitations for example not applicable for aerial vehicle or handheld camera. It will also fail when the ground is not visible such as frames in Fig 6.4 of KITTI 07. The front dynamic car occludes the ground for a long time and that's why many ground-based approaches fail or perform poorly on KITTI 07.

6.7.2 Dynamic object SLAM

We also test the algorithm on the dynamic car sequences on KITTI datasets shown in Fig 6.10(a). We select some raw sequences with more dynamic objects observed over long time shown in Table 6.4. The full name of the sequences is "2011_0926_00xx". The first four sequences also correspond to Seq $3, 4, 5, 18$ in KITTI tracking dataset. Ground truth object annotations are available for all or some frames and ground truth camera poses are also provided by GPS/INS. In these sequences, all cars are assumed to be dynamic, therefore, there is no need of motion segmentation to detect whether the object is dynamic or not and benefits the fair comparison of different dynamic SLAM algorithms.

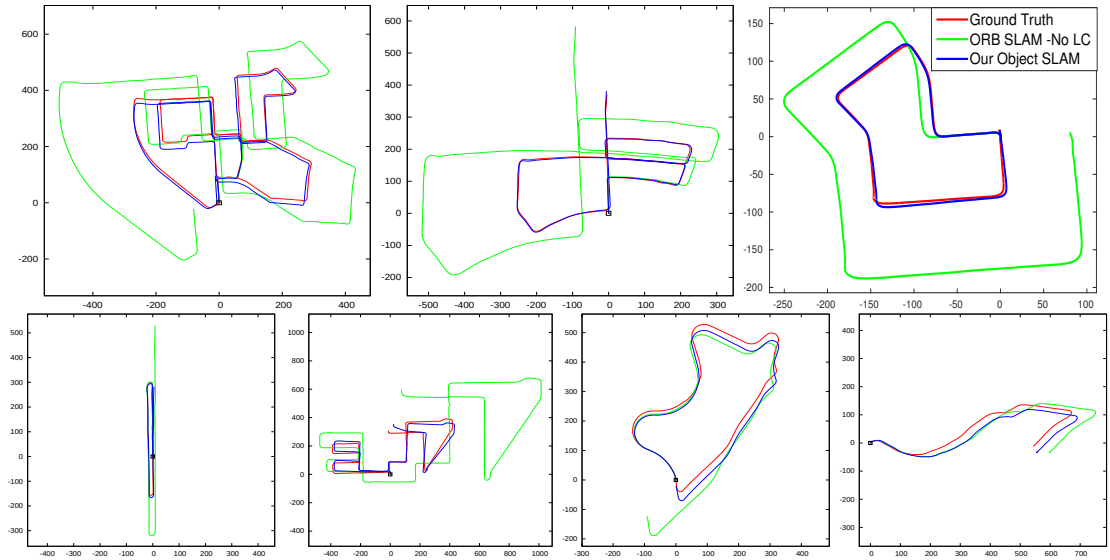


Figure 6.9: Our object SLAM on KITTI odometry dataset without loop closure and constant ground height assumption. Red is ground truth. Blue is our object SLAM. Green is ORB point SLAM without loop closure. Objects can reduce monocular scale drift and improve pose estimation accuracy.

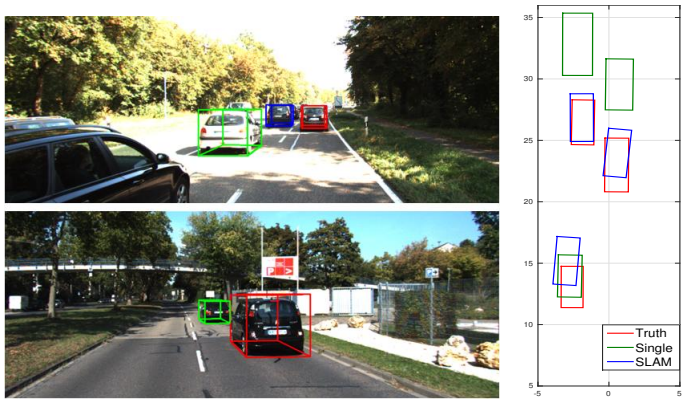
Qualitative results

Some single image detection examples are shown in Fig 6.10(a). Fig 6.10(b) shows the top view of the first image of Fig 6.10(a). For the two distant front cars, even though the 2D image cuboid detection looks good but it actually has large 3D distance error. This is because only cars' back faces are observable causing ill-constraint single image detection. After multi-view dynamic object BA, the blue optimized object matches better with the ground truth mostly due to motion model constraints. However it can sometimes decrease the accuracy for the bottom object. Some possible reasons are due to the noisy 2D and 3D object detection especially for the close objects. The constant motion assumption may also cause errors when the vehicle accelerates or decelerates. Fig 6.10(c) shows all the dynamic object's history poses as well as camera poses. The objects' trajectories are smooth due to the motion model constraints.

Fig 6.11 shows the velocity estimation of one of objects in on Seq 0047 data. We can see that the computed ground truth object velocity also changes with time therefore the piecewise constant velocity motion explained in Sec 6.6 is reasonable. With a monocular camera, the proposed algorithm can roughly estimate the object's absolute velocity.

Quantitative results

Since there is no current monocular SLAM utilizing dynamic objects to affect camera pose estimation, we thus directly compare with the state-of-the-art featured based ORB SLAM. Though it already has some modules to detect dynamic points as outliers based on reprojection error, to compare with ORB SLAM fairly, we directly remove features lying in the dynamic object areas and report its pose estimation result. From Table 6.4, we can see our method can improve the



(a) (b)

(c)

Figure 6.10: Dynamic object SLAM result on KITTI. (a) Samples frames of single image cuboid detection (b) Top view comparison of 3D detection before and after dynamic BA optimization. (c) Camera and object poses over long sequence. The red curve starting from left is the camera's trajectory. Other curves with rectangle markers represent the dynamic object's trajectory.

Table 6.4: Dynamic Object Detection and Camera Pose Estimation on KITTI Raw Sequence

Seq		13	14	15	04	56	32	Mean
Object	Single view	0.41	0.11	0.42	0.11	**0.54**	**0.34**	0.32
3D IoU	Ours	**0.51**	**0.28**	**0.44**	**0.42**	0.42	0.26	**0.39**
Trans	ORB -No LC	2.34	11.5	2.4	2.31	8.45	**2.76**	4.96
error(m)	Ours	**0.99**	**7.62**	**1.94**	**1.50**	**5.39**	3.07	**3.42**

camera pose estimation on most sequences especially when objects can be observed and tracked over many consecutive frames for example in the first four datasets. This is because with more observations, objects' velocity and dynamic points can be estimated more accurately and thus have more effect on the camera pose estimation, while in the last two sequences, objects are usually observed by only a few frames.

We also compare the 3D object localization with other monocular methods shown in Table 6.5. The most similar one to us is [125] which utilized semantic and geometric costs to optimize object locations but their approach assumed the camera poses are already solved and fixed. We utilize the same metric in [125] to measure the relative object depth error in each frame and the number is directly taken from the paper. From the table, our method outperforms others on most object sequences. In the sequence 56, the relative depth error is only 0.8%.

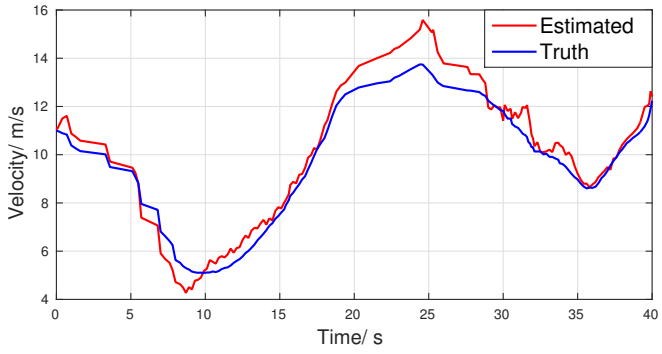


Figure 6.11: Dynamic object velocity estimation on KITTI Seq 0047. The SLAM algorithm is based on piecewise constant motion model and can correctly estimate the moving object's velocity using a monocular camera.

Table 6.5: Dynamic Object Localization Comparison on KITTI Raw Sequence

Seq		04				47				56	Mean
Obj. ID		1	2	3	6	0	4	9	12	0	
No. Frames		91	251	284	169	170	96	94	637	293	
Depth	[133]	**4.1**	6.8	5.3	7.3	9.6	11.4	7.1	10.5	5.5	7.9
error	[125]	6.0	**5.6**	4.9	5.9	**5.9**	12.5	7.0	8.2	6.0	6.8
(%)	Ours	5.1	11.1	**1.8**	**2.5**	9.3	**3.8**	**6.6**	**2.8**	**0.8**	**4.9**

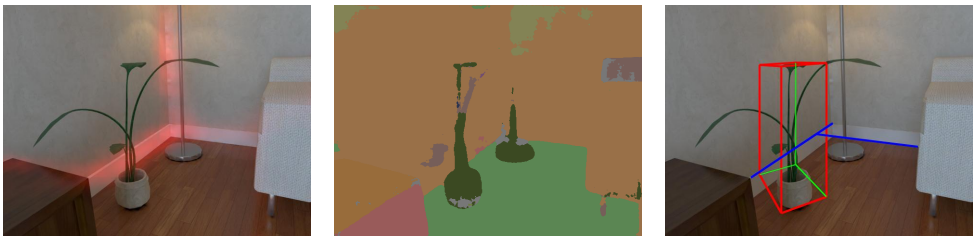
6.7.3 Object and Plane SLAM

We also evaluate then object and plane SLAM performance on both public datasets including ICL-NUIM [130], TAMU Indoor [134], TUM mono [135], and our collected datasets by KinectV2 sensor.

Qualitative Results

A sample frame of ICL sequence is shown in Fig 6.12. The left and middle images show the raw image overlayed by layout prediction and the semantic segmentation. Both of them have noise and CRF optimization in Fig .6.12(c) shows a roughly correct 3D model but it cannot fully detect the occluded wall segments. After the multi-view SLAM optimization, the algorithm is able to build a more consistent and complete map shown in Fig. 6.2.

More 3D mapping and camera pose estimation in different datasets and environment configurations are shown in Fig 6.13. After BA, objects and planes' locations are more accurate compared to the single view detection and most objects lie inside the room. Note that not all objects are mapped because the 2D object detector might miss some and SLAM might also treat some of them as outliers due to inconsistent observations. In some scenarios such as the top left, our algorithm cannot detect the full wall plane due to severe object occlusions. To improve the visualization robustness, if there is not enough map point observed in some region of a plane polygon, no dense pixels will be projected, shown as the void segments on the wall surface in



(a) (b) (c)

Figure 6.12: (a) Layout prediction score map [53] (b) Semantic segmentation by [104] (c) Our CRF optimized cuboid object and wall planes. It cannot detect the occluded wall surface while multi-view SLAM can build a complete map in Fig. 6.2.

Table 6.6: Absolute Camera Translation Error on Various Datasets

Method	ORB SLAM-NO LC [1]	Ours
ICL living 0	3.08	**0.8**
ICL living 2	3.25	**2.06**
ICL living 3	**5.36**	5.38
ICL office 0	6.23	**5.93**
ICL office 2	5.00	**2.63**
Tamu corridor	3.87	**0.97**
Our room 1	0.15	**0.05**
Our corridor 1	2.25	**0.30**
Our corridor 2	2.93	**0.24**
Our corridor 3	1.84	**0.49**

the middle image.

Quantitative Results

We then show the quantitative camera pose comparison with ORB SLAM and DSO. For datasets in Table 6.6, the initial map of both ORB and ours is scaled by the truth initial camera height. Then we can directly evaluate the absolute translation error without aligning the pose in scale, to show that object and planes can improve the pose estimation and reduce monocular drift. Each algorithm runs in each sequence for 5 times and the mean error is reported here. From the table, we can see that in most of the scenarios, the added objects and planes landmark constraints improve the camera pose estimation. There are two main reasons for this. One is that even though we disable explicit loop closure, due to object and plane's long-range visibility properties, the algorithm may still associate with the old plane landmark to reduce the final drift. The second reason is the depth initialization of features especially in large camera rotations. Due to the strict outlier rejection and robust BA optimization, even if it doesn't improve the result, it won't seriously damage the system.

Table 6.7: Pose Alignment Error on TUM-mono Dataset

Method	ORB-No LC[1]	DSO [2]	Ours
Corridor 36	1.81	4.01	**0.94**
Room 37	0.60	0.55	**0.35**
Corridor 38	23.9	**0.55**	7.65

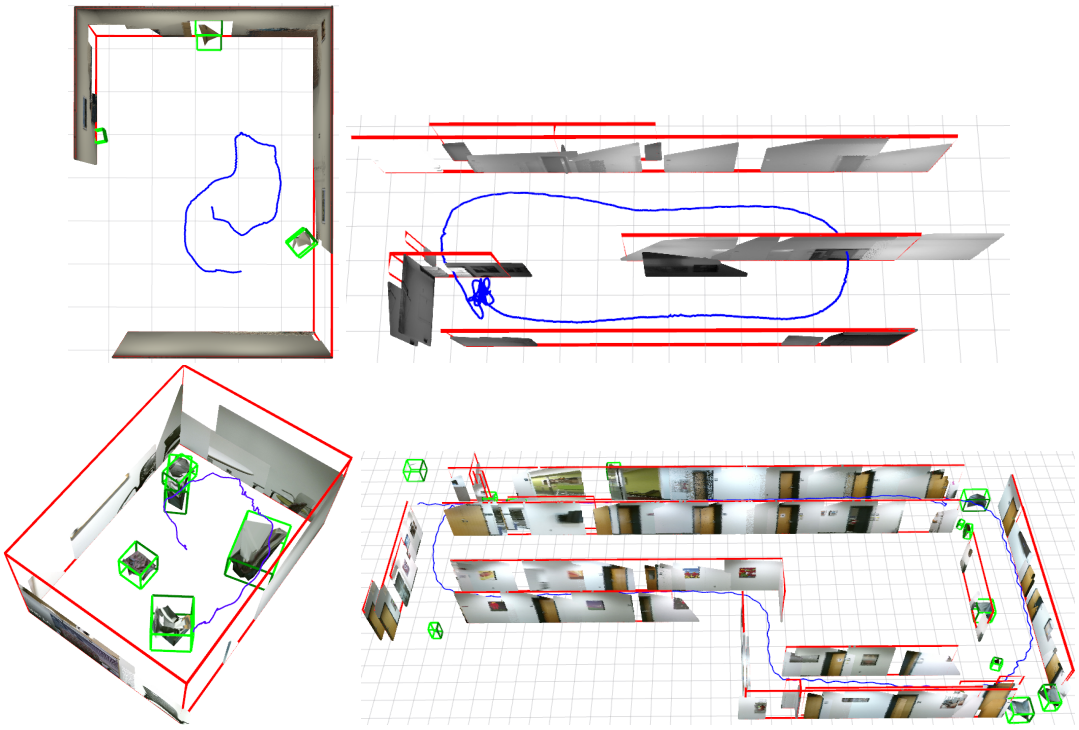


Figure 6.13: More dense mapping results with objects and planes. (top) ICL-NUIM office 2, TUM-mono 36. (middle) Our collected room and long corridor.

For TUM mono data in Table 6.7, no truth camera height is available thus we evaluate the monocular scale alignment error [2]. Results of DSO and ORB are taken from the supplementary material of DSO. Our semantic SLAM can work robustly in these challenging datasets even though there is large camera rotation and sometimes the camera may be upside down. In the cluttered dataset such as Room 37, there are only a few planes with a few observed frames thus our algorithm almost reduces to point SLAM and achieves similar results. In Corridor 38, our algorithm and ORB SLAM are much worse compared to DSO because there are many areas of white walls with few feature points which are difficult for feature based SLAM.

6.7.4 Time analysis

Finally, we provide the computational analysis of our system. The experiments are carried out on Intel i7-4790 CPU at 4.0 GHz, and Nvidia 980 Ti GPU. GPU is used for 2D object detector and

Table 6.8: Average runtime of different system components

Tasks		Runtime (mSec)
Single-image Preprocessing	2D object detection, MS-CNN KITTI	383
	2D object detection, YOLO Indoor	17.5
	2D semantic segmentation, SegNet	71.5
	Edge detection	12.1
Outdoor SLAM KITTI-07	Tracking Thread (per frame)	33.0
	Point BA (per keyframe)	182.7
	Point + Static Object	194.5
	Point + Dynamic Object BA	365.2
Indoor SLAM ICL room	Tracking Thread (per frame)	15.0
	Point BA (per keyframe)	49.5
	Point + Object + Plane BA	105.6

semantic segmentation. All SLAM part is implemented in C++ on CPU. As shown in Table 6.8, there are several pre-processing steps for our single image 3D object and plane detection such as 2D object detection, semantic segmentation and so on. The CNN algorithms we used cannot run in real time but they depend on the model complexity and GPU power. There are also some recent fast and lightweight CNNs that could run in real time for example [136].

We then evaluate the time usage of different SLAM components. We separate outdoor and indoor datasets due to quite different image size and textures. Outdoor data is measured on KITTI 07 sequence (10Hz raw data) shown in Fig 6.1(b) and indoor data is for ICL-NUIM living room2 dataset (30Hz raw data) shown in Fig 6.2. The tracking thread includes the ORB point feature detection and camera pose tracking for each frame. It can run in real time from the table. The bundle adjustment (BA) map optimization occurs when a new keyframe is created therefore doesn't need to run in real-time. We show the time usage of BA when different types of landmarks exist. In the static environment, adding objects into the system only increases the optimization by 7%. This is also reasonable as there are only a few objects in the local map. For the indoor environments, compared to point only BA, plane landmarks double the optimization time because the point-plane constraints are applied to most of the points, bringing in more costs to optimize. Another reason relates to the implementation of g2o. Since there are different types of edges with different dimensions such as camera-point, point-plane, we cannot pre-allocate the solver matrix dimensions. For the dynamic environments, there are also more variables in SLAM such as object poses in each frame and dynamic map points, therefore, the time also increases by twice.

6.8 Conclusions

In this work, we propose a monocular 3D object detection and SLAM method without prior object models, in both static and dynamic environments. In addition, we also propose the first monocular SLAM and dense mapping algorithm with points, objects and planes as SLAM land-

marks. We demonstrate for the first time, that semantic object and plane understanding and geometric SLAM can benefit each other in one unified framework.

For the BA optimization, several new measurement functions are designed for planes and objects landmarks. We also propose new association methods, strict outlier detection and robust optimization to improve the robustness. Some practical plane boundary update methods are proposed for monocular camera. Objects and planes can provide long-range geometric and scale constraints for camera pose estimation. In turn, SLAM also provides camera pose initialization for detecting and refining 3D objects. For the dynamic scenarios, we also show that with the new measurement constraints, the moving object and point can also improve the camera pose estimation through the tightly coupled optimization.

We evaluate the SLAM algorithm in various public indoor and outdoor datasets and achieves better pose estimation and mapping quality than the state-of-the-art. Meanwhile, it also improves the 3D object detection accuracy. The object SLAM achieves the best camera pose estimation on KITTI odometry benchmark. In the future, more general planes in addition to wall planes need to be considered to produce a denser and more complete map. More complete scene understanding can be integrated with SLAM optimization.

In the future, more general planes in addition to wall planes need to be considered to produce a denser and more complete map. Object surfaces also need to be reconstructed.

Chapter 7

Semantic 3D Occupancy Mapping

In this chapter, we propose a filtering based online and incremental semantic mapping system. Different from the tightly SLAM optimization in Chapter 6, the SLAM mapping is decoupled of the semantic labelling. We first build a 3D map then directly transfer the 2D semantic information to 3D map followed by a hierarchical graphical optimization. Instead of using object and planes, grid is used here for more general representation.

7.1 Introduction

3D semantic mapping is important for many robot applications such as autonomous navigation and robot interaction. Robots not only need to build 3D geometric maps of the environments to avoid obstacles, but also need to recognize objects and scenes for high-level tasks. For example, autonomous vehicles need to locate and also classify vehicles and pedestrians in 3D space to keep safe. However, there are also many challenges with this task. Instead of offline batch optimization, it should be processed incrementally in real time rates and computation time should be independent of the size of environments.

The problem is composed of two parts: geometric reconstruction, and semantic segmentation. For 3D reconstruction, there has been a large amount of research on visual simultaneous localization and mapping (SLAM). The map can be composed of different geometric elements such as points [1], planes[33], and grid voxels [137] etc. For semantic segmentation, current research usually focuses on image or video segmentation for 2D pixel labeling. With the popularity of convolutional neural networks (CNN), the performance of 2D segmentation has greatly improved. However, 2D semantic reasoning is still not accurate in the case of occlusion and shadowing.

Recently, there has also been some work on semantic 3D reconstruction [63] [62]. However, most of the existing approaches suffer from a variety of limitations. For example, they cannot run incrementally in real time and cannot adapt to large scale scenarios. Some system can achieve real-time rates with GPU acceleration [62]. Different sensors can be used to accomplish the task such as RGBD cameras [61], however, they can only work indoors in small workspaces, therefore we choose stereo cameras due to their wide applicability to both indoor and outdoor environments.

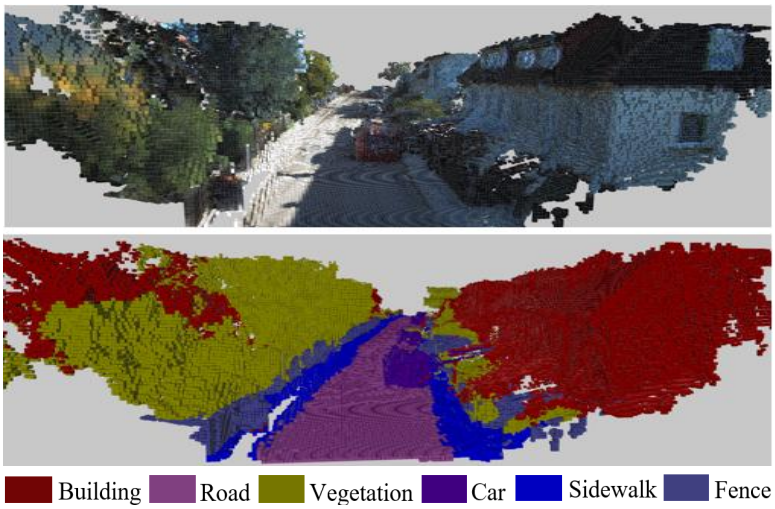


Figure 7.1: (top) Geometric and (bottom) semantic 3D reconstruction. Our system can incrementally create semantic map of large scale environments using scrolling occupancy grids in (near) real time. We utilize the latest CNN and build a novel hierarchical CRF model to optimize the label.

In this work, we utilize CNN model to compute pixel label distributions from 2D image and transfer it to 3D grid space. We then propose a Conditional Random Field (CRF) model with higher order cliques to enforce semantic consistency among grids. The clique is generated through superpixels. We develop an efficient filter based mean field approximation inference for this hierarchical CRF. To be applicable to large-scale environments and to achieve real-time computation, a scrolling occupancy grid is built to represent the world which is memory and computationally bounded. In all, our main contributions are:

- Propose a (near) real-time incremental semantic 3D mapping system for large-scale environments using a scrolling occupancy grid
- Improve the segmentation accuracy by 10% over the state-of-the art systems on the KITTI dataset
- Develop a filter based mean filed inference for high order CRFs with robust P^n potts model by transforming it into a hierarchical pairwise model

The paper is organized as follows. We first provides some literature review. The system contains two main parts: geometric mapping in Section 7.2 and semantic mapping in Section 7.3. Results and conclusions are presented in Section 7.4 and Section 7.5.

Closely Related Work

In this section, we first give an overview of the geometric 3D reconstruction and semantic reconstruction at the time of publication. A more complete review about it can be found in Chapter 2. We then provide in-depth review of CRF optimization.

Table 7.1: Comparison with other system

Method	Images only	Incremental	High-order CRF	Real-time
Sengupta [60]	✓			
Hermans [61]		✓		✓
Kundu [63]	✓	✓	✓	
Sengupta [141]	✓	✓	✓	
Vinnet [62]	✓	✓		✓
Zhao [142]			✓	–
Li [143]	✓	✓		✓
Ours	✓	✓	✓	✓

3D reconstruction There are many different algorithms for geometric 3D mapping in recent years such as ORB SLAM [1], and DSO [2], which can achieve impressive results in normal environments. Their maps are usually composed of points, lines or planes. Since points are continuous in space, it is time-consuming and even intractable to perform inference each point's label. A typical approach is to divide the space into discrete grids such as an occupancy grid [138] [139] and octomap [140], which have already been used in many dense and semantic mapping algorithms [62].

Semantic reconstruction A summary of the relevant semantic 3D mapping work is provided in Table 7.1. A straightforward solution is to directly transfer the 2D image label to 3D by back-projection [60] without further 3D optimization. Hermans *et al.* [61] propose to optimize 3D label through a dense pairwise CRF. Vineet *et at.* [62] apply the same CRF model to large scale stereo 3D reconstruction and achieve real-time rates by GPU. Recently, more complicated high order CRF models are also used for 3D reasoning. Kundu *et al.* [63] model voxel's occupancy and label in one unified CRF with high order ray factors. The super-voxels in octomap [141] or 2D superpixel [142] can also be used to form high order potential.

CRF Inference Recently, *dense CRFs* [84] have become a popular tool for semantic segmentation and have been applied to many systems [62] [142]. Dense CRFs can model long range relationships compared to a basic neighbour connected CRF model. More complicated CRF with *high order potentials* can be used to encourage label consistency within one region. Since exact inferences for CRFs is generally intractable, many *approximation algorithms* have been developed, such as variants of belief propagation or mean field approximation for dense CRFs [84]. Vineet *et al.* [144] extend this inference to CRFs with a P^n Potts model, applied to 3D semantic mapping system[142].

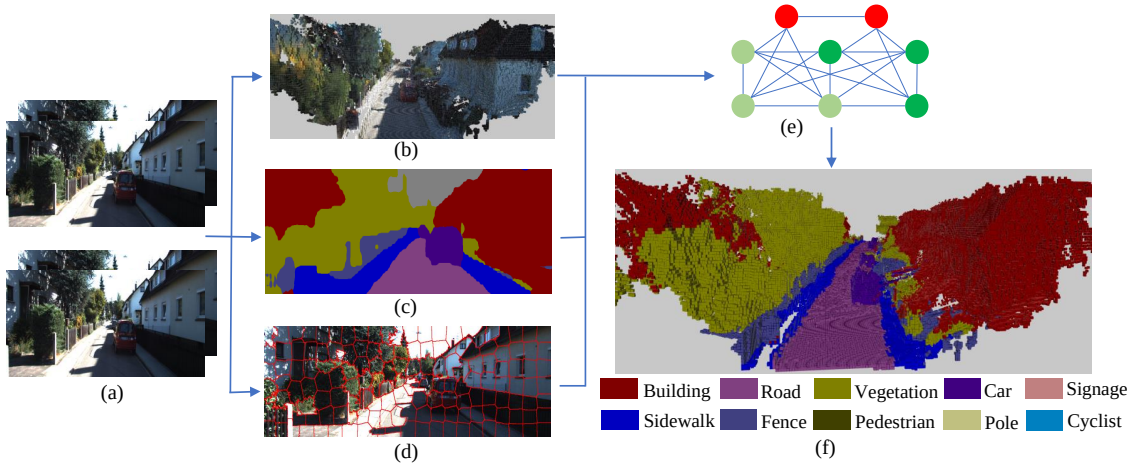


Figure 7.2: Overview of our system. (a) Input stereo image pairs. (b) Geometric 3D reconstruction using ORB SLAM [1] and occupancy mapping [139][138]. (c) 2D semantic segmentation using CNN [145], treated as the CRF unary potential. (d) Superpixel clique generation using SLIC [146] to enforce label consistency within a region, to calculate CRF high order potential. (e) Proposed hierarchical CRF model. We develop efficient mean field inference for hierarchical CRF in 3D grid space. (f) Final semantic mapping after 3D CRF optimization.

7.2 Geometric Mapping

The system diagram is shown in Fig. 7.2. The input to our system is a series of stereo pair images and the output is an incrementally constructed 3D semantic grid map. We break the problem into two steps: geometric 3D mapping (this section) and semantic 3D labelling (next section).

7.2.1 3D mapping

We first build a 3D geometric map from stereo image pairs which contains three steps: stereo depth estimation, camera pose estimation, and 3D grid mapping.

In order to achieve a dense 3D mapping, we need to accurately estimate stereo disparity with high density. We adopt ELAS [147] which forms a triangulation on a set of support points to robustly estimate disparity in low-texture areas. We then utilize stereo ORB SLAM [1] to estimate 6DoF camera pose. It detects the ORB feature points and then minimizes the reprojection error across different views.

Since ORB SLAM only generates a sparse map of feature points which cannot be used for dense mapping, we project each frame's estimated dense disparity to 3D space. In order to fuse the observations across different views, we change the point cloud map into 3D occupancy grids [138]. Each grid stores the probability of being occupied and is updated incrementally through ray tracing based on stereo depth measurement. To keep memory and computation efficient, the occupancy map keeps a fixed dimension and moves along with the camera. If the occupancy value exceeds a threshold, the grid will be considered as occupied and considered for the latter CRF optimization. Note that since *sky* has no valid depth, we ignore the classified *sky* pixels

during occupancy mapping.

7.2.2 Color and Label fusion

Standard occupancy grid maps only store the occupancy value, however in our case, we also need to store the color and label distribution for latter CRF optimization. Since each grid can be observed by different pixels in different frames, we need to fuse the observation before optimization. For color fusion, we directly take the mean of different color observation in different views. For the label fusion, we follow the standard Bayes' update rule similar to that in occupancy mapping. Denote the label probability distribution of a grid at time t as x_t and the measurements till now as $z_{1:t}$. For the new combing image z_t, we first perform 2D semantic segmentation then update the 3D grid's label based on Bayes rule:

$$\begin{aligned} p(x_t|z_{1:t}) &= \frac{p(x_t|z_t)p(z_t)}{p(x_t)}\frac{p(x_{t-1}|z_{1:t-1})}{p(z_t|z_{1:t-1})} \\ &\approx \frac{1}{Z}p(x_t|z_t)p(x_{t-1}|z_{1:t-1}) \end{aligned} \tag{7.1}$$

To derive the second row, we can assume the class prior $p(x_t)$ as a constant and group $p(z_t)/p(z_t|z_{1:t-1})$ as the normalization constant. So for each incoming image, the label probability fusion is simply multiplication followed by normalization. The updated $p(x_t)$ will be further be optimized through CRF.

7.3 Hierarchical Semantic Mapping

After building the 3D grid map, we utilize CRFs to jointly optimize each grid's label. We propose a general hierarchical CRF model with a robust P^N potential and develop an efficient inference algorithm for it. Note that this CRF model can also be used for other problems such as 2D segmentation.

We start by defining each grid's label as a random variable x_i taking a label from a finite label set $\mathcal{L} = \{l_1, l_2, ..., l_k\}$ and the joint label configuration of all N grids as $\mathbf{x} \in \mathcal{L}^N$. Then the joint probability and Gibbs energy is defined:

$$P(\mathbf{x}|\mathbf{D}) = \frac{1}{Z(\mathbf{D})}\exp(-E(\mathbf{x}|\mathbf{D})) \tag{7.2}$$

$$E(\mathbf{x}|\mathbf{D}) = \sum_i \psi_i^U(x_i) + \sum_{i<j} \psi_{ij}^P(x_i, x_j) + \psi_{\mathbf{c}}^{HO}(\mathbf{x_c}) \tag{7.3}$$

where $P(\mathbf{x}|\mathbf{D})$ is the posterior probability of configuration $\mathbf{x}$ given a grid data $\mathbf{D}$, $Z(\mathbf{D})$ is the partial function for normalization. ψ_i^U and ψ_{ij}^P are the unary and pairwise potential energy. $\psi_{\mathbf{c}}^{HO}$ is the high order energy formed by all cliques $\mathbf{x_c}$. We will explain these three potentials in more details. Now the CRF inference problem of maximizing $P(\mathbf{x}|\mathbf{D})$ changes to find $\mathbf{x}^* = \arg\min_{\mathbf{x}} E(\mathbf{x}|\mathbf{D})$.

7.3.1 Unary potential

Unary $\psi_i^U(x_i)$ represents the cost of a grid taking label x_i, which can also be treated as the prior distribution of variables. It is computed by the negative logarithm of the prior probability:

$$\psi_i^U(x_i) = -\log p(x_i) \tag{7.4}$$

where $p(x_i)$ is the fused label distribution from Equation 7.2 in Section 7.2.2. There are many approaches on 2D semantic segmentation for example TextonBoost [148] and more recent CNN [83][110]. We adopt the dilated CNN [145] simply because it directly provides the trained model. Note that it doesn't contain post optimization such as CRF so the prediction might contain discontinuity and inconsistency between pixels as shown in Fig. 7.2(c).

7.3.2 Pairwise potential

The pairwise potential $\psi_{ij}^P(x_i, x_j)$ is adopted from [84], defined as a combination of gaussian kernels:

$$\psi_{ij}^P(x_i, x_j) = \mu(x_i, x_j) \sum_{m=1}^{K} w^{(m)} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) \tag{7.5}$$

where $\mu(x_i, x_j)$ is the label compatibility function. We use the simple Potts model $\mu(x_i, x_j) = [x_i \neq x_j]$, which introduces a penalty for nodes assigned different labels. Each $k^{(m)}$ is a Gaussian kernel depending on feature vectors $\mathbf{f}$. We currently use two kernel potentials defined based on the color and position of 3D grid:

$$k(\mathbf{f}_i, \mathbf{f}_j) = w_1 \exp\Big(-\frac{|p_i - p_j|}{2\theta_\alpha^2} - \frac{|I_i - I_j|}{2\theta_\beta^2} \Big) + w_2 \exp\Big(-\frac{|p_i - p_j|}{2\theta_\gamma^2} \Big) \tag{7.6}$$

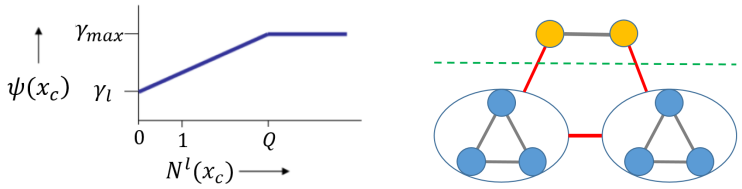
There are also some other features in 3D space for example surface normal [62]. However, our occupancy grid is not dense and smooth enough to compute high quality surface normals.

7.3.3 Hierarchical CRF Model

The above dense pairwise potential lacks the ability to express more complicated and meaningful constraints. For example in a 2D image, pixels within a homogeneous region are likely to have the same label. Thus, we design high-order potentials $\psi_c^{HO}(x_c)$ to represent these constraints.

There are different models for high order potentials. The P^n Potts model [149] rigidly enforces the nodes within a clique to take the same label which might be wrong due to inaccurate clique segmentation. Kohli *et al.* [150] then propose a Robust P^n model whose cost is dependent on the number of variables taking the dominant label shown in Fig. 7.3(a). It could be treated as a soft version of P^n Potts model. Robust P^n is further shown to be equivalent to the minimization of a hierarchical pairwise graph with new added auxiliary variable y_c representing the clique's label [151] shown as the yellow nodes in Fig. 7.3(b). The robust P^n potential for this clique is then defined as:

$$\psi_c^{HO}(x_c) = \min_{y_c} \psi_c(x_c, y_c) = \min_{y_c} \big(\psi_c(y_c) + \sum_{i \in c} \psi_{ci}(y_c, x_i) \big) \tag{7.7}$$



(a) (b)

Figure 7.3: (a) Robust P^n Model cost for clique c from [149]. (b) Our hierarchical CRF model by representing clique as an yellow auxiliary node. The auxiliary clique node has dense pairwise connections between each other and also has connections to its children nodes. Best viewed in color.

where $\psi_c(y_c)$ represents the auxiliary clique variable's unary cost. A separate classifier could be trained to compute it or we can simply use the mean of its children nodes' unary. $\psi_{ci}(y_c, x_i)$ encourages the consistency between clique variable and its children nodes shown as the edge between yellow nodes and blue nodes in Fig. 7.3(b), defined as:

$$\psi_{ci}(y_c, x_i) = \begin{cases} 0 & \text{if } y_c = x_i \\ k_c^l & \text{Otherwise} \end{cases} \tag{7.8}$$

We further extend it to model the relationships between all auxiliary variables y as a gaussian pairwise potential defined in Section 7.3.2, shown as the edges between yellow nodes. It basically encourages the label consistency between similar segment cliques. The clique variable's feature is computed by the mean of RGB and position of all its children nodes.

In summary, let V represents the set of low-level grid variables and $\mathcal{S}$ be the high-level clique variables. Then the total energy $E(\mathbf{x})$ in Equation 7.3 is transformed into:

$$E(\mathbf{x}) = \sum_{i \in \mathcal{V}} \psi_i(x_i) + \sum_{i,j \in \mathcal{V}} \psi_{ij}(x_i, x_j) + \min_{\mathbf{y}} \Big(\sum_{c \in \mathcal{S}} \psi_c(x_c, y_c) + \sum_{c,d \in \mathcal{S}} \psi_{cd}(y_c, y_d) \Big) \tag{7.9}$$

where the first row defines the unary and pairwise potentials of low-level grid variables and the second row defines the high order potential of cliques and between cliques. Now this model is only composed of unary and pairwise terms but with more auxiliary variables.

7.3.4 Mean Field Inference for Hierarchical CRFs

Algorithm Description

Mean field inference for CRFs with P^n potts model has been addressed by [144]. In this section, we develop efficient inference for our hierarchical CRF model with Robust P^n model.

We approximate the target distribution $P(\mathbf{x})$ using $Q(\mathbf{x})$ with the form $Q(\mathbf{x}) = \prod_i Q_i(x_i)$, namely all variables are marginally independent. During each iteration, we first update the distribution of high-level clique variables y_c using Equation 7.10 and find the MAP label assignment for them then update low-level grid nodes x_i using Equation 7.11. The two mean field update rules are as follows:

$$Q^{t+1}(y_c = l) = \frac{1}{Z_c} \exp\left(-\psi_c(y_c) - \sum_{d \in \mathcal{S}, d \neq c} Q_d(y_d)\psi_{cd}(y_c, y_d) - \sum_{i \in c} Q_i(x_i)\psi_{ci}(y_c, x_i) \right) \quad (7.10)$$

$$Q^{t+1}(x_i = l) = \frac{1}{Z_i} \exp\left(-\psi_i(x_i) - \sum_{j \in \mathcal{V}, j \neq i} Q_j(x_j)\psi_{ij}(x_i, x_j) - Q_c(y_c)\psi_{ci}(y_c, x_i) \right) \quad (7.11)$$

Complexity Analysis

There are two main parts in the previous equation: dense pairwise terms $\psi_{cd}(y_c, y_d)$, $\psi_{ij}(x_i, x_j)$, and clique-children terms $\psi_{ci}(y_c, x_i)$.

For the dense pairwise parts, as shown by Krähenbühl [84], using the technique of Gaussian convolutions and permutohedral lattice, the time complexity of the Potts model is $\mathcal{O}(KNL)$, where K is the number of kernels, N is total grid number and L is the labels. So in our case the computation would be $\mathcal{O}(KN_gL + KN_cL)$, where $N_g = |V|$, $N_c = |S|$ are the number of low-level grids and high-level cliques respectively.

For the clique-children terms, we need to visit each low-level grid to check its label compatibility with the clique variable. In our setting, each grid only has at most one parent clique so the computation complexity would be $\mathcal{O}(N_g)$.

In all, the total time complexity is $\mathcal{O}(KN_gL + KN_cL + N_g) \approx \mathcal{O}(KN_gL)$ as clique number N_c is usually much smaller than grid number N_g. Therefore, it has the same algorithm complexity with dense pairwise CRF in theory, which is linear in the number of grid N_g.

7.4 Experiments

7.4.1 Dataset and implementation

We evaluate our system on two labelled datasets from KITTI [107]: Sengupta [60] containing 25 test images from sequence 15, and Kundu [63] containing 40 test images mainly from sequence 05. They are also used in other state-of-the-art semantic 3D mapping system [62] [141] [60]. In total, there are 11 object classes including building, road, car and so on.

As explained in Section 7.3.1, our unary prediction comes from the dilation CNN [145] which is trained on another sequence of KITTI dataset. Fine-tuning on the experimental dataset could also be used to further improve the accuracy. We modify the scrolling occupancy grid library [152] to maintain a 3D map. The map size is $25 \times 25 \times 8m^3$ around the camera with grid resolution $0.1m^3$. Larger grid volume and finer grid resolution could improve the performance but the computation time also increases quickly. Grid cells outside of the bounding area will be removed to save memory and computation for online updates, but could also be stored in memory so as to create a final complete 3D map.

We use the SLIC algorithm to generate around 150 superpixels per image, shown in Fig. 7.2(d). We transfer the pixel-superpixel membership to 3D space by back projecting pixels to the corresponding 3D grids.

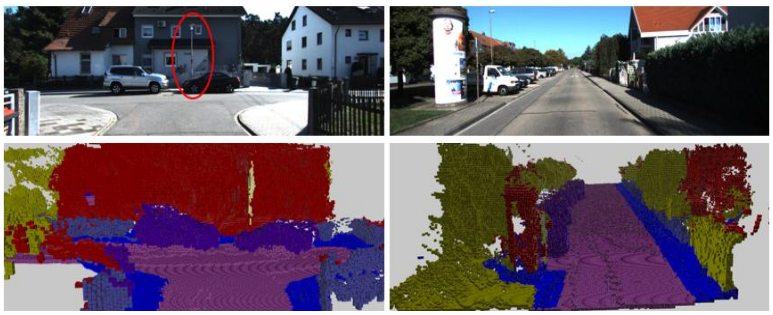


Figure 7.4: Qualitative results of our 3D semantic mapping. First row: input images. Second row: semantic reconstruction scene. A thin pole is marked in red and also constructed in the map. In the second image, the building and car are clearly recognized.

To evaluate the segmentation accuracy, we project 3D labelled grid maps onto the camera image plane, ignoring grids that are too far from the camera. We choose 40 meters as the threshold, instead of the 25 meters in [62]. Though further grids have larger position uncertainty and reduce the segmentation accuracy, they can improve the mapping density and can also be beneficial for other applications.

7.4.2 Qualitative Results

We first present some qualitative results of semantic 3D mapping in Fig. 7.4. Our approach can successively recognize and reconstruct classes of general objects, and even thin objects such as poles in spite of the shadows and textureless surfaces. The top view of the whole sequence (keep all grids in memory) and more examples are shown in Fig. 7.6.

In another example of Fig. 7.5, we demonstrate the advantage of 3D segmentation compared to 2D. In Fig. 7.5(a), due to the dark shadow of trees, *fence* in the red eclipse area has similar intensity and texture with its surrounding trees thus it is very difficult to label them only from 2D image shown in Fig. 7.5(b) and Fig. 7.5(c). However, in 3D space, *fence* is continuous and has different shape and position compared to trees and roads thus it can be correctly labelled using 3D optimization shown in Fig. 7.5(d).

7.4.3 Quantitative Results

In this section, we quantitatively evaluate and compare the segmentation accuracy with other approaches.

Comparison with 3D system

As mentioned in Section 7.1, different systems take different 3D geometric mapping approaches such as voxel-hashing [62], octomap [141] and our occupancy grid, therefore a common approach for comparison is to project the 3D map onto image planes. We adopt the standard metric

Table 7.2: Comparison of 2D/3D approaches on KITTI Sengupta (seq15) dataset

		Method	Building	Vegetation	Car	Road	Fence	Sidewalk	Pole	Average	Global
Accuracy	2D	Texton[148]	97.0	93.4	93.9	98.3	48.5	**91.3**	49.3	81.7	88.4
		CNN	**98.5**	**97.6**	**96.5**	**98.4**	**77.8**	87.9	**51.1**	**86.8**	**94.2**
	3D	Sengupta[60]	96.1	86.9	88.5	97.8	46.1	86.5	38.2	77.2	85.0
		Valentin[153]	96.4	85.4	76.8	96.9	42.7	78.5	39.3	73.7	84.5
		Sengupta[141]	89.1	81.2	72.5	97.0	45.7	73.4	3.30	66.0	78.3
		Vineet[62]	97.2	94.1	94.1	98.7	47.8	91.8	51.4	82.2	—
		Our before CRF	98.2	98.5	**96.3**	**99.5**	81.8	90.2	57.1	88.9	95.1
		Our after CRF	**98.2**	**98.7**	95.5	98.7	**84.7**	**93.8**	**66.3**	**90.9**	**95.7**
IoU	2D	Texton[148]	86.1	82.8	78.0	**94.3**	47.5	73.4	39.5	71.7	
		CNN	**93.3**	**89.8**	**94.1**	93.4	**76.4**	**80.0**	**44.1**	**81.7**	
	3D	Sengupta[60]	83.8	74.3	63.5	96.3	45.2	68.4	28.9	65.7	
		Valentin[153]	82.1	73.4	67.2	91.5	40.6	62.1	25.9	63.2	
		Sengupta[141]	73.8	65.2	55.8	87.8	43.7	49.1	1.9	53.9	
		Vineet[62]	88.3	83.2	79.5	94.7	46.3	73.8	41.7	72.5	
		Our before CRF	94.2	90.8	**95.4**	95.2	79.9	84.8	54.2	85.2	
		Our after CRF	**95.4**	**91.0**	94.6	**96.6**	**81.1**	**90.0**	**61.5**	**87.6**	

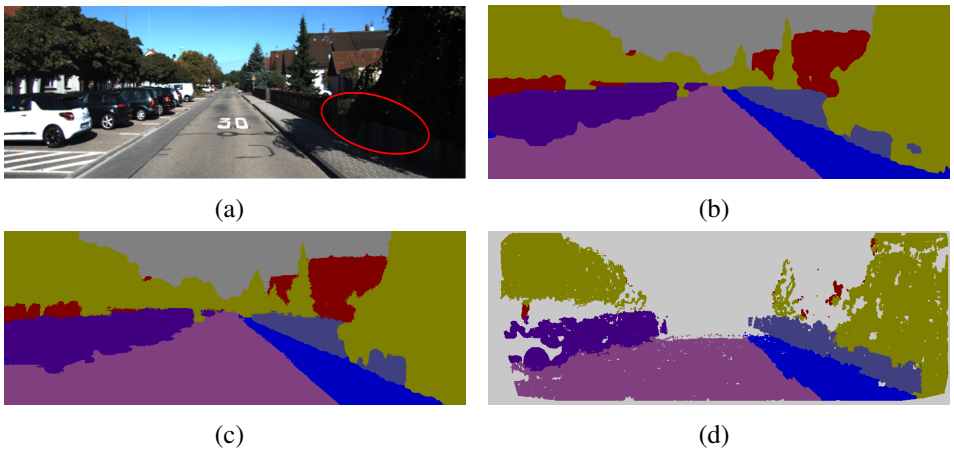


Figure 7.5: Examples demonstrating the advantage of 3D optimization. (a) Raw RGB image (b) 2D CNN prediction (c) 2D CRF optimization. *fence* on the bottom right is still wrongly labelled due to the dark shadow of trees. (d) 3D CRF optimization result (back-projection onto image plane). Gray area means no projections. Fence is now correctly labelled.

of (class) pixel accuracy defined as TP/(TP+FP) and (class) intersection over union (IoU) defined as TP/(TP+FP+FN). T/F P/N represents true/false positive/negative.

Comparison results are shown in Table 7.2. *Global* represents the overall pixel accuracy and *Average* represents mean class accuracy or IoU. '—' indicates number not provided. In total, there are 11 class in the original dataset and we select 7 common class appearing in other systems for comparison. The results of other work are taken from the paper directly. Since [141] and [153] also evaluate the *signage* class which is ignored here, we recompute the metric *Average* of their methods. Metric *Global* cannot be re-computed as it depends on the actual distribution but it only has about 0.1% variation because *signage* occupies a very small area.

From the table, our method greatly outperforms other 3D systems in all categories. For the class *fence*, there is a significant accuracy increase of 36.3% over the state-of-the-art, and for the class *pole*, we achieve a 10.3% improvement. Overall, the mean class IoU is increased by 15% and global pixel accuracy is increased by 10%.

There are two main reasons why our approach outperforms existing systems. First, we use the latest the 2D CNN semantic segmentation as CRF unary shown as row 'CNN' in the table, while existing approaches all use TextonBoost [148] shown as row 'Texton'. CNN is much more accurate than TextonBoost nearly in all categories. Even using the simple label fusion in Section 7.2.2 without further CRF optimization, the result in row *Our before CRF* is already more accurate than other systems. Second, we propose a new hierarchical CRF model to optimize 3D grid labels which further improves the mean IoU by 2.4% and global accuracy by 0.6%. In some class such as *Sidewalk* and *Pole*, 3D CRF optimization has about 6% improvement of IoU, which also matches the example in Fig. 7.5.

Table 7.3: Comparison of 2D CRF model on KITTI Kundu (seq05) dataset (IoU)

Class	CNN	Dense CRF	P^N Potts	Our hier
Building	83.8	85.4	84.5	**86.6**
Sky	87.6	90.3	89.7	**91.8**
Road	90.0	**90.2**	90.0	90.1
Vegetation	83.0	83.9	82.9	**83.9**
Sidewalk	74.4	**74.7**	74.5	74.3
Car	72.5	**73.8**	73.0	73.1
Sign	23.1	29.3	24.7	**40.6**
Fence	69.5	**70.4**	69.5	70.1
Pole	**33.6**	30.6	32.4	23.5
Mean IoU	68.6	69.6	69.0	**70.3**
F.W IoU	81.5	82.6	81.9	**82.7**
Global Acc	89.9	90.5	90.1	**90.6**

Comparison of different CRF model

In this section, we compare different CRF models to demonstrate the advantage of our proposed hierarchical model. As mentioned in Section 7.3, our CRF model can also be applied to other optimization problems not limited to 3D mapping. So we evaluate the CRF on a 2D semantic segmentation task because it is not affected by other factors such as 3D reconstruction error, grid map resolution etc. We choose a more diverse Kundu dataset [63] for evaluation.

We compare with two other popular CRF models: dense CRF [84] and P^N Potts model [144], also adopted in other 3D mapping system. All the CRF models utilize the CNN prediction as unary cost then optimize for 5 iterations. Some qualitative comparison are shown in Fig. 7.7 while quantitative result is shown Table 7.3. F.W. IoU stands for frequency weighed IoU. We can see that our model has higher IoU and global accuracy. For class *sign*, we increase 10.5% IoU compared to other models. This is mainly because that *sign* is composed of triangular or square plates which usually form separate superpixels, thus suitable for our CRF model. However, some thin long objects such as *Pole* usually break to different superpixels, making it difficult to optimize in 2D. However, in 3D space shown in previous Table 7.2, our CRF model can still work well because even *pole* is broken into multiple superpixels, its 3D position is quite different from its surroundings, therefore can still easily be classified.

7.4.4 Time analysis

We also provide time analysis of different components in our system. Except for the CNN unary prediction, all other computation are implemented on desktop CPU Intel i7-4790. CNN prediction time is not included here as it depends on the model complexity and GPU power. In fact, there are also some fast and lightweight CNNs that could even run in real time on embedded system [136].

There are five main components of our system shown in Table 7.4. The first three parts utilize

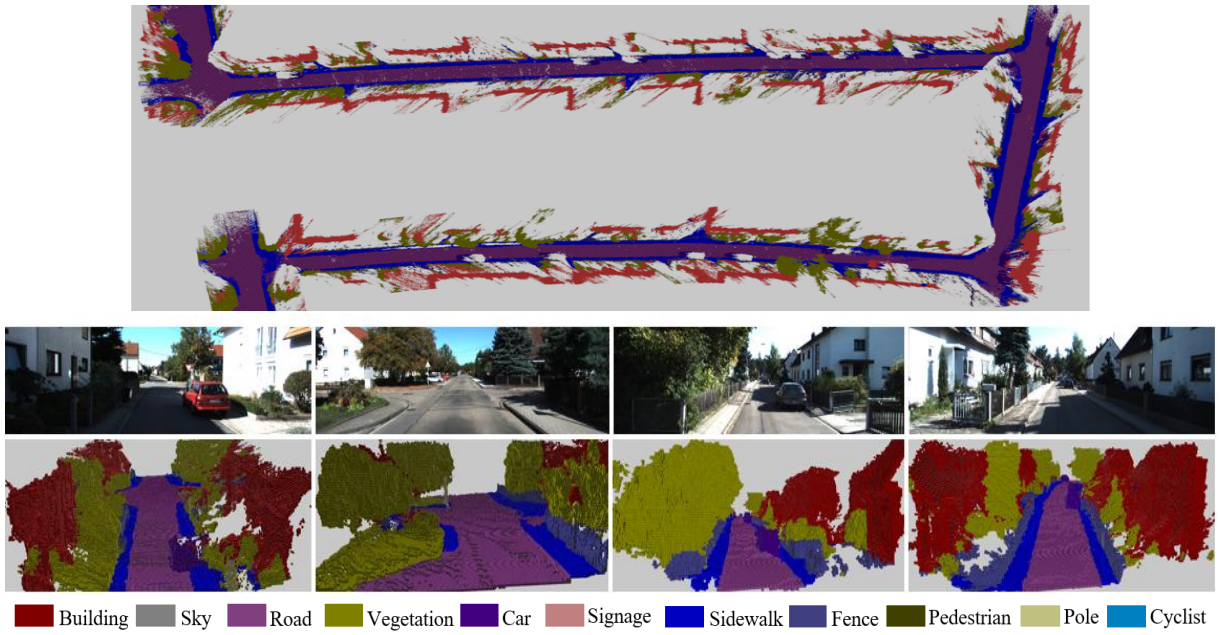


Figure 7.6: Visualization of 3D semantic mapping. (a) top view of 3D mapping over long sequence (850 images) of KITTI sequence 5. It demonstrates that our algorithm work well in large scale environments. (b) More 3D reconstruction examples with different scenarios.

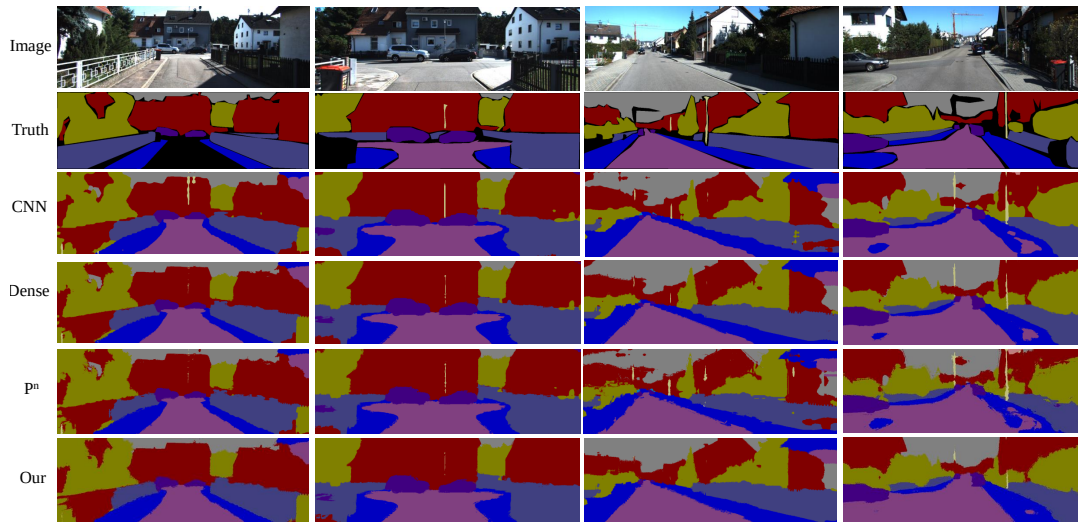


Figure 7.7: Comparison of different CRF models on 2D semantic segmentation task in KITTI sequence 5. *Dense* stands for dense CRF. P^N stands for high order CRF with P^N Potts model. The last line is our proposed hierarchical CRF with robust P^N model. The black area in truth image is not labelled. Our method on the bottom row has better overall performance for example the sky region in the third image and road on the last image.

Table 7.4: Time analysis of our algorithm

Method	Time(s)
Dense Disparity	0.416
State Estimation	0.102
Superpixel Generation	0.115
Occupancy Mapping	0.295
CRF w/o Hierarchical	0.53/0.38

the existing public libraries and can further be computed in parallel threads to improve speed. From the label fusion in Section 7.2.2, the grid probability distribution from previous frames is updated and further optimized by the current frame's observation. Since each frame only updates a small part of the 3D grid map, we only need to optimize for a few iterations until convergence. Through our test, even one single mean-field update is enough to produce good results. In Table 7.2, the result is also reported where CRF iteration is set to one.

From the table, in the current settings, the grid optimization can run at 2Hz for hierarchical CRF or 2.5Hz without high order CRF potential, slower compared to the image frame rates (10Hz). As analyzed in Section 7.3.4, the computation $\mathcal{O}(KNL)$ is linear in the number of grids and label. There are several possible ways to improve the speed. Firstly, we can reduce the grid size or grid resolution to reduce the number of variables N in the CRF optimization. However, this might increase the 3D reconstruction error and may not qualify for other needs. Secondly, reduce the number of labels L. If we only optimize for 7 main classes instead of current 11 classes, the speed can be almost doubled. Thirdly, GPU could also be used for parallel mean field updates which has been used in [62]. Lastly, we actually don't need to process each frame as adjacent frames may appear quite similar and have small affect the 3D map. Through our test, by processing every four frames to satisfy real-time rates, the global accuracy only reduces from 95.7 to 95.3 and IoU reduces from 87.6 to 86.3.

7.5 Conclusions

In this chapter, we presented a 3D online semantic mapping system using stereo cameras in (near) real time. Scrolling occupancy grid maps are used to represent the world and are able to adapt to large-scale environments with bounded computation and memory. We first utilize the latest 2D CNN segmentation as prior prediction then further optimize grid labels through a hierarchical CRF model. Superpixels are utilized to enforce smoothness. Efficient mean field inference for the high order CRF with robust P^N potential is achieved by transforming it into a hierarchical pairwise CRF. Experiments on the KITTI dataset demonstrate that our approach outperforms the state-of-the-art approaches in terms of segmentation accuracy by 10%.

Our algorithm can be used by many applications such as virtual interactions and robot navigation. In the future, we are interested in creating more abstract and high level representation of the environments such as planes and objects. Faster implementation on GPU could also be explored.

Chapter 8

Direct Monocular Odometry Using Lines

We have explored different SLAM landmarks such as points, objects and planes in the previous chapters. In this chapter, we utilize another important geometric feature: line, to improve the visual odometry performance. Photometric and geometric error are combined to improve state estimation accuracy in some low texture environments.

8.1 Introduction

Visual odometry (VO) and Simultaneous localization and mapping (SLAM) have become popular topics in recent years due to their wide application in robot navigation, 3D reconstruction, and virtual reality. Different sensors can be used such as RGB-D cameras [154], stereo cameras [27] and lasers, which could provide depth information for each frame, making it easier for state estimation and mapping. However for some applications such as weight constrained micro aerial vehicles [138], monocular cameras are more widely used due to their small size and low cost. Therefore, in this work, we are aiming at the more challenging monocular VO.

There are typically two categories of VO and vSLAM approaches: (1) feature based methods such as PTAM [24] and ORB SLAM [1]. They rely on feature point extraction and matching to create sparse 3D map used for pose estimation by minimizing re-projection geometric error. (2) Recently, direct method [118][25] also becomes popular. It directly operates on the raw pixel intensity by minimizing photometric error without feature extraction. These two methods both have their advantages. Reprojection geometric error of keypoints is typically more robust to image noise and large geometric distortions and movement. Direct method on the other hand, exploits much more image information and can create dense or semi-dense maps.

In this paper, we utilize points and edges to combine the advantages of the above two approaches. Edge is another important feature apart from points. It has been used for stereo [155] and RGB-D VO [156], but receives less attention in monocular VO. The detection of edges is less sensitive to lighting changes by nature. For example, in a homogeneous environment of Fig. 8.1, direct method using points only may not work robustly due to small image gradient, but we can still detect many edges shown in blue in the figure which could be used for state estimation and mapping. In our system, we maintain a semi-depth map for the keyframe's high gradient pixels as in many direct VO methods [118]. We also detect and match edges for each frame.

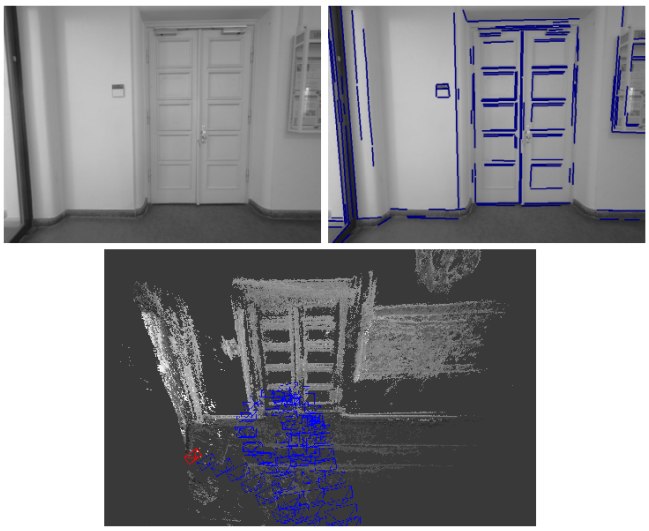


Figure 8.1: Tracking and 3D reconstruction on TUM mono dataset using our edge based visual odometry. The top image shows the homogeneous wall surface with low image gradients, which is challenging for VO only minimizing photometric error. However, edge shown in blue can still be detected to improve the tracking and mapping performance.

Then in the tracking part, we jointly optimize both photometric error and geometric error to the corresponding edge if it has. In the mapping part, edges could also be used to guide and speed up the stereo search and also improve depth map quality by edge regularizing. By doing this, the proposed VO can increase the accuracy of state estimation and also create a good semi-dense map. We demonstrate this through various experiments.

In summary, our main contributions are:

- A real-time monocular visual odometry algorithm incorporating points and edges, especially suitable for texture-less environments.
- Provide an uncertainty analysis and probabilistic fusion of points and lines in tracking and mapping.
- Develop analytical edge based regularization
- Outperform or comparable to existing direct VO in many datasets.

In the following section, we discuss related work. In Section 8.2, we provide the problem formulation. Tracking and mapping using points and edges are presented in Section 8.3 and Section 8.4 respectively, which also include probabilistic uncertainty analysis of different observation model. In Section 8.5, we provide experimental comparison with the state-of-art algorithm. Finally, conclusion and future work is discussed in Section 8.6

Related Work

Our algorithm utilizes edges to combine feature based and direct VO. We briefly introduce these three aspects. Some related and more complete review can be found in Chapter 2.

Feature based VO There have been many feature point based VO and SLAM, for example LibVISO [27] and ORB SLAM [1]. They first extract image features then track or match them across images. The camera pose is estimated by solving the PnP (Perspective N-Point Projection) problem to minimize geometric error which is more robust to image noise and has a large convergence basin [1] [2]. The drawback is that the created map is usually sparse. A separate direct mapping algorithm is required to get a semi-dense map [157].

Direct VO In recent years, direct method [34] also becomes popular. It optimizes the geometry directly on the image intensities without any feature extraction so it can work in some texture-less environments with few keypoints. It has been used for real-time application of different sensors for example DVO for RGB-D cameras [158] and LSD SLAM for monocular cameras [25]. The core idea is to maintain a semi-dense map for keyframes then minimize the photometric error which is a highly non-convex function thus it requires good initial guess for the optimization. In between direct and feature based methods, SVO combines direct alignment and feature points and can be used for high frame rate cameras.

Edge based VO Edges are another important feature apart from points especially in man-made environments. Edges are more robust to lighting changes and preserve more information compared to single points. Line-based bundle adjustment has been used in SLAM or SfM [28] [159] which are computationally expensive and require at least three frames for effective optimization. Line-based VO without bundle adjustment has recently been used for stereo cameras [160] [155] and RGB-D [161] [156] and monocular cameras [29][162]. Kuse *et al.* minimized the geometric error to its nearby edges pixels through distance transform [156] which might cause wrong matching due to false detected edges and broken edges while our line segment matching could greatly reduce the error. Some works only minimize geometric error of two edge endpoints [155] [161] which may generate large error for monocular cameras due to inaccurate depth estimation.

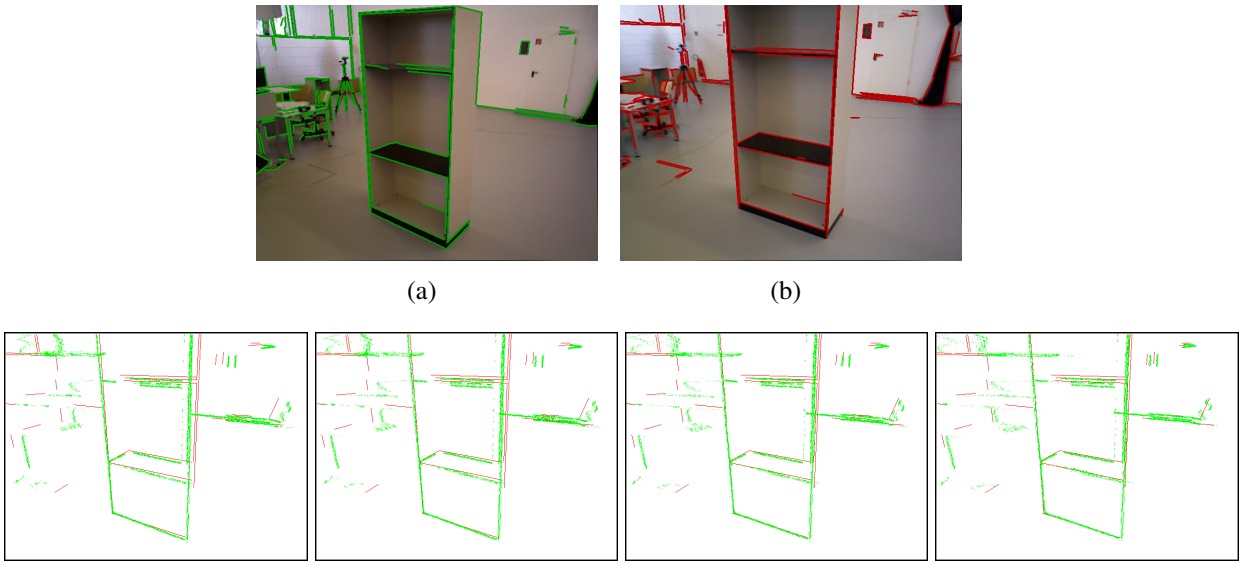
8.2 Problem Description

8.2.1 System Overview

Our algorithm is a frame to keyframe monocular VO. We maintain a semi-depth map for the high gradient pixels in the keyframe. Then for each incoming new frame, there are three steps. First, detect line segments and match them with the keyframe's edges. The second step is camera pose tracking. We minimize a combination of pixel photometric error and geometric reprojection error if the pixel belongs to an edge. Lastly, we update the depth map through variable baseline stereo. Edges are used to speed up the stereo search for those edge pixels and also improve reconstruction through an efficient 3D line regularization.

8.2.2 Notations

We denote an intensity image as $I : \Omega \subset \mathbb{R}^2 \mapsto \mathbb{R}$, where Ω represents the image domain. We keep a per-pixel inverse depth map for a reference keyframe $D : \Omega \subset \mathbb{R}^2 \mapsto \mathbb{R}^+$ and inverse



(a) (b)

(c)

Figure 8.2: Tracking iterations for two images in TUM *fr3/cabinet_big* dataset. (a) reference frame with detected edges (b) current frame with detected edges. Two frames are 41 frames apart (about 1.4s). (c) Re-projected pixels on current frame during optimization iterations corresponding to $1, 4, 9, 20$. We can see that the re-projected edge pixels in green gradually align with the true edges in red. Best viewed in color.

depth variance $V : \Omega \mapsto \mathbb{R}^+$.

The camera projection function is defined as $\pi : \mathbb{R}^3 \mapsto \mathbb{R}^2$, which projects a camera-centered 3D point onto image plane. The inverse projection function is then $\pi^{-1} : (\mathbb{R}^2, \mathbb{R}) \mapsto \mathbb{R}^3$ which back-projects an image pixel to the 3D space given its depth.

The transformation between the current frame and reference keyframe is defined by a rigid transformation $T \in SE(3)$. For an efficient optimization of T, we use the minimal manifold representation by elements of the Lie-algebra $\xi \in \mathfrak{se}(3)$ [163], which is expressed by twist $\xi = (\mathbf{t}; \mathbf{w})^T \in \mathbb{R}^6$. $\mathbf{t} \in \mathbb{R}^3$ is the translation component and $\mathbf{w} \in \mathbb{R}^3$ is the rotation component, which can form rotation matrix by the corresponding exponential map: $R(\mathbf{w}) = \exp([\mathbf{w}]_\times) \in SO(3)$.

A warping function is defined as $\tau : \Omega_1 \times \mathbb{R} \times \mathbb{R}^6 \mapsto \Omega_2$. It takes the parameters of a pixel $x \in \Omega_1$ in the first image I_1, its depth d and relative camera transformation ξ, then returns the re-projection point in second image I_2. Internally, it first back-projects x to 3D point by π^{-1}, transforms it using ξ, then projects to another frame using π.

An edge is represented by L in 3D space and l in 2D image plane. All the edge pixels in an image are defined as $M : \mathbb{R}^2 \mapsto l$ which maps a pixel to its edge. Most of the edge pixels M belong to the high gradient pixel set Ω.

8.3 Tracking

8.3.1 Overview

In the tracking thread, the depth map D_{ref} of the reference frame I_{ref} is assumed to be fixed. The current image I is aligned by minimization of the photometric residual $r(\xi)$ and line re-projection geometric error $g(\xi)$ corresponding to two observation model: photometric intensity observation and edge position observations. It can be formulated as the following non-linear least squares problem:

$$E(\xi) = \sum_{i \in \Omega} r_i(\xi)^T \Sigma_{r_i}^{-1} r_i(\xi) + \sum_{j \in M} g_j(\xi)^T \Sigma_{g_j}^{-1} g_j(\xi) \tag{8.1}$$

where photometric error r_i is defined by [118] [34]:

$$r_i = I_{ref}(x_i) - I(\tau(x_i, D_{ref}(x_i), \xi)) \tag{8.2}$$

g_j is the re-projection error of pixel x_i to its corresponding line l_j (homogeneous line representation):

$$g_j = l_j^T \, \widehat{\tau}(x_i, D_{ref}(x_i), \xi) \tag{8.3}$$

where $\widehat{\tau}()$ is the homogeneous coordinate operation. This term is only used for the pixels of edges in I_{ref} which also have a matching edge in I. Σ_r and Σ_g represents the uncertainty of two errors correspondingly.

The energy function Equation (8.1) is minimized through iterative Gauss-Newton optimization. For iteration n, the small update is:

$$\delta\xi^n = -(J^T W J)^{-1} J^T W \mathbf{E}(\xi^n) \tag{8.4}$$

Where $\mathbf{E}$ is the stacked error vectors composed of two parts: $\mathbf{E} = (r_1, ..., r_n, g_1, ..., g_m)^T$. J is the the Jacobian of $\mathbf{E}$ *wrt.* ξ. W is the weight matrix computed from uncertainty Σ^{-1}. A tracking illustration is shown in Fig 8.2.

8.3.2 Tracking uncertainty analysis

Combining different types of error terms in Equation (8.1) increases the robustness and accuracy of pose estimation. The weights of different terms are proportional to the inverse of the error variance Σ_r and Σ_g computed from the observation models. Here, we provide an analysis of Σ_g. Photometric error uncertainty Σ_r has been analysed in [118].

In the general case, the uncertainty of the output of a function $f(x)$ propagated from the input uncertainty is expressed by:

$$\Sigma_f \approx J_f \Sigma_x J_f^T \tag{8.5}$$

where J_f is the Jacobian of f *wrt.* x.

In our case, as defined in Equation (8.3), the pixel re-projection error to line is a function of line equation l_j and re-projected point $x_i' = \widehat{\tau}(x_i, D_{ref}(x_i), \xi)$. Line equation is computed by cross product of two line endpoints $l_j = p_1 \times p_2$. We can assume that the uncertainties Σ_p of

end point positions p_1 and p_2 is bi-dimensional Gaussians with $\sigma = 1$. Then we can use rule in Equation (8.5) to compute the uncertainties of line equation coefficients l_j. It basically implies that longer line has smaller line fitting uncertainties.

We can then similarly compute the variance of re-projection point $x'_i = \widehat{\tau}(x_i, D_{ref}(x_i), \xi)$. It is a function of pixel depth $D_{ref}(x_i)$ with variance $V_{ref}(x_i)$. The final re-projection error covariance is a combination of the two uncertainty sources:

$$\Sigma_{g_j} = l_j^T \Sigma_{x'_i} l_j + x_i'^T \Sigma_{l_j} x'_i \tag{8.6}$$

8.4 Mapping

8.4.1 Overview

In the mapping thread, the depth map D_{ref} of reference frame is updated through stereo triangulation in inverse depth filtering framework [118] followed by line regularization to improve the accuracy. The camera pose is assumed to be fixed in this step. The cost function for depth optimization is defined as follows:

$$E(D) = \sum_i r_i(d)^T \Sigma_{r_i}^{-1} r_i(d) + \sum_j G_j(d)^T \Sigma_{G_j}^{-1} G_j(d) \tag{8.7}$$

where $r_i(d)$ is the stereo matching photometric error. SSD error over image patches is used to improve robustness. For a line l_j, we want its pixels to also form a line in 3D space after back-projection, so G_j is edge regularization cost representing the distance of edge pixel's 3D point to 3D line. The regularization technique is also used in other dense mapping algorithms [34] [68]. If only the first term r_i is used [118], all the pixels are independent of each other and therefore could search independently along the epipolar line to find the matching pixel. Regularization term G_j makes the depth of pixels on one edge correlate with each other and is typically solved by an iterative alternating optimization through duality principles [34]. However, it requires much heavier computation. Instead, we optimize for r_i and G_j in two stages more efficiently.

8.4.2 Stereo match with Lines

For the pixels not on the edge or pixels on an edge which does not have a matching edge, we perform an exhaustive search for the stereo matching pixel by minimizing SSD error [118]. The depth interval for searching is limited by $d + 2\sigma_d$, where d and σ_d is the depth mean and standard deviation.

For the pixels with a matched edge, the re-projected points should lie on the matched edge as well as its epipolar line so we can directly compute their intersection as the matching point. We can also directly do line triangulation in Fig. 8.3 to compute all pixel's depth together. If the camera transform of current frame I wrt. I_{ref} is $R \in SO(3), \mathbf{t} \in \mathbb{R}^3$, then the 3D line L can be represented the intersection of two back-projected plane [114]:

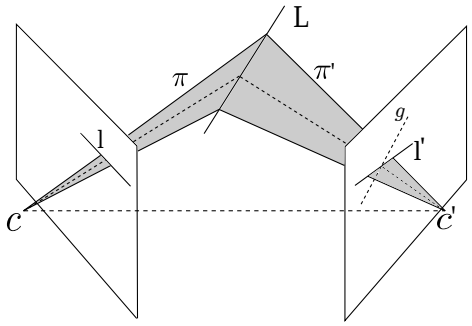


Figure 8.3: Line triangulation. 3D line L could be computed by the intersection of two back-projected planes π, π'. For each pixel on l, its stereo matching point is the intersection of epipolar line g and matched edge l'. The triangulated point also lies on 3D line L. Modified from [114].

$$L = \begin{bmatrix} \pi_1^T \\ \pi_2^T \end{bmatrix} = \begin{bmatrix} l_1^T K & 0 \\ l_2^T K R & l_2^T K \mathbf{t} \end{bmatrix} \tag{8.8}$$

where l_1 and l_2 are the line equation in I_{ref} and I respectively. K is the intrinsic camera parameter. Then for each pixel, we can compute the intersection of the back-projected ray with L to get its depth.

For the degenerated case where epipolar line and matched edge are (nearly) parallel, we cannot compute the 3D line accurately by plane intersection. Instead, we use the exhaustive search along the epipolar line to find the matching pixel with minimal SSD error.

8.4.3 Line matching uncertainty analysis

The uncertainty of intensity based stereo searching along epipolar line has been analysed in [118]. Here we include the analysis of edge based stereo matching error. For each edge pixel in I_{ref}, denote its epipolar line in I as g and its matched edge as l then the matching pixel is the intersection of g and l. These two lines both have positioning error ϵ_l and ϵ_g, and finally cause a disparity error ϵ_λ shown in Fig. 8.4. The edge uncertainty ϵ_l is already analyzed in Section 8.3.2 which is directly related to the edge length. ϵ_λ is large when g and l are nearly parallel. Mathematically we have:

$$\epsilon_\lambda = \epsilon_l / \sin(\theta) + \epsilon_g \cot(\theta) \tag{8.9}$$

where θ is the angle between line l and g.

From error propagation rule in Equation (8.5), we can compute the variance of the disparity error:

$$\sigma_\lambda^2 = \sigma_l^2 / \sin^2(\theta) + \sigma_g^2 \cot^2(\theta) \tag{8.10}$$

Using the approximation that inverse depth d is proportional to disparity λ, we can calculate the observation variance of d using Equation (8.5). It can then be used to update the pixel's depth variance in a standard EKF filtering [118].

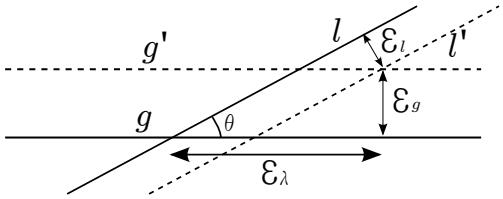


Figure 8.4: Disparity error using line matching. l is the edge where the matched pixel should lie. g is the epipolar line. Due to a small positioning error ϵ_g, g is shifted to g'. The same with l' of positioning error ϵ_l. The final resulting disparity error is ϵ_λ.

8.4.4 3D Line regularization

Depth map regularization is important for monocular mapping approaches to improve the depth estimation accuracy. After the depth map EKF update in Section 8.4.3, the pixels on a 2D edge may not correspond to a line in 3D space, therefore, we need to fit lines in 3D space and update a pixel's depth. 3D weighted line fitting is recently addressed in RGB-D line based odometry [161] which utilizes Levenberg-Marquardt iterative optimization to find the best 3D line. Here we propose a fast and analytical solution to the weighted 3D line fitting problem.

Since the 3D points are back-projected from the same 2D edge, they should lie on the same plane G from projective geometry. We can create anther coordinate frame F whose x, y axis lie on the plane G. The transformed point on the new coordinate frame is denoted as p'. We first use RANSAC to select a set of inlier 2D points. The metric for RANSAC is Mahalanobis distance, which is a weighted pixel to line Euclidean distance considering the uncertainty:

$$d_{\text{mah}} = \min_{q' \in l'} (p' - q')^T \Sigma_{p'}^{-1} (p' - q') \tag{8.11}$$

where $q' \in l'$ indicates a point lying on line l' in frame F. d_{mah} could be computed analytically by taking the derivative *wrt.* q' and setting to zero. More details could be found in [161].

After RANSAC, we can find the largest consensus set of points p'_i, $i = 1, ..., n$. This becomes a 2D weighted line fitting problem and we want to find the best line L^* so that:

$$L^* = \min_L \sum_i \delta(p'_i)^T \Sigma_{p'_i}^{-1} \delta(p'_i) \tag{8.12}$$

where $\delta(p'_i)$ is distance of point p'_i to line L along y axis. It is an approximation of point to line distance but could lead to a closed form solution. Stack all points p'_i coordinates as $[\mathbf{X}, \mathbf{Y}]$ (after subtracting from mean) and weight matrix as W which can be approximated as original image pixels' covariance. Then the line model under consideration is $\mathbf{Y} = \mathbf{X}\beta + \epsilon$, where β is line coefficients, and ϵ is assumed to be normally distributed vector of noise. The MLE optimal line under Gaussian noise is:

$$\widehat{\beta} = arg \min_\beta \sum_i \epsilon_i^2 = (\mathbf{X}^T W \mathbf{X})^{-1} \mathbf{X}^T W \mathbf{Y} \tag{8.13}$$

We can then transform the optimal line L^* in coordinate frame F back to the original camera optical frame and determine the pixel depth on the line.

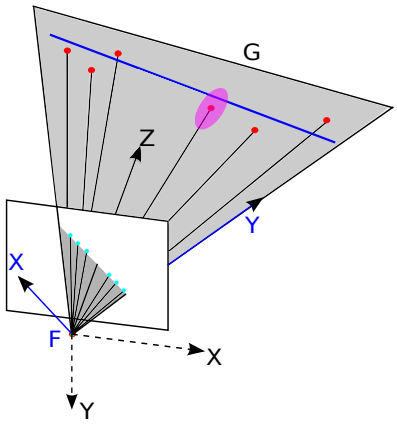


Figure 8.5: 3D line depth regularization. We first un-project pixel to 3D shown as red dots on the 3D plane G. The pink ellipse shows the uncertainty of 3D point. We can transform 3D points to a coordinate frame F lying on the grey plane. The axis are X, Y in blue. Then we can use RANSAC to analytically compute a weighted least square line instead of iterative optimization [161]. Image modified from [161]. Best viewed in color.

8.5 Experiments

8.5.1 Implementation

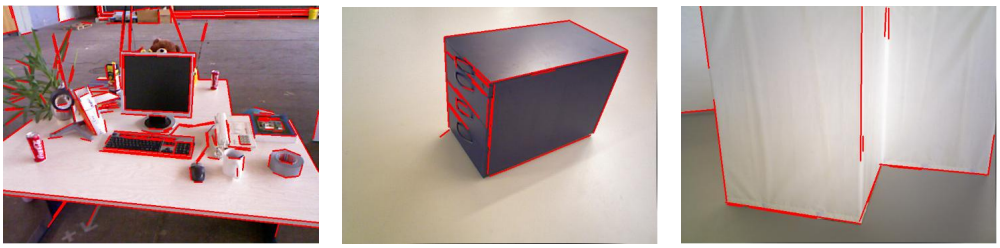
1. Edge detection and matching: We use the public line segment detection algorithm [164]. To improve the tracking accuracy, we adopt a coarse-to-fine approach using two pyramid levels with a scale factor of two. Due to the uncertainty of line detection algorithm, one complete line can sometimes break into multiple segments so we explicitly merge two lines whose angles and distance are very close within a threshold. After that, we need to remove very short line segments which may have large line fitting error. To speed up the line merging, lines are assigned to different bucket grids indexed by the middle points of an edge and the orientation of it. Then we only need to consider possible merging within the same and nearby bucket.

We then compute the LBD descriptor [165] for each line and match them across images. Bucket technique is also utilized to speed up the matching. Finally, line tracing is performed to find all pixels on an edge. We find that the system becomes more robust and accurate if we expand the line for one pixel possibly because more pixels are involved by the line constraints in tracking and mapping.

2. Keyframe-based VO: our approach doesn't have the bundle adjustment of points and lines in SLAM and SfM framework but could be extended to improve the performance. Camera tracking, line matching and stereo mapping are implemented only between the current frame and keyframe.

8.5.2 Results

In this section, we test our algorithm on various public datasets including TUM RGBD [119], TUM mono [135] and ICL-NUIM [130]. We mainly compare with the state of art monocular



(a) (b) (c)

Figure 8.6: Example images in TUM datasets with varying textures. (a) fr2/desk, (b) fr3/cabinet, (c) fr3/notex-far. ORB SLAM performs worse on (b) and (c) as there are fewer features points. Our algorithm can still utilize the matched edge features to improve the state estimation.

direct SDVO [118] and feature based ORB SLAM [1]. We also provide some comparison with edge based VO [29] [162] in some datasets where the result is provided. For ORB SLAM, we turn off the loop closing thread, but still keep local and global bundle adjustment (BA) to detect incremental loop-closures while our algorithm and SDVO are VO algorithm without BA. We use the relative position error metric (RPE) by Strum *et al* [119].

$$E_i = (Q_i^{-1} Q_{i+\delta})^{-1} (B_i^{-1} B_{i+\delta}) \tag{8.14}$$

where $Q_i \in SE(3)$ is the sequence of ground truth poses and $B_i \in SE(3)$ is the estimated pose. Scale is estimated to best align the trajectory.

Qualitative results

We choose TUM *mono/38* [135] for VO and mapping visualization shown in Fig. 8.1. It mainly contains homogeneous white surfaces but there are still many edges that could be utilized. Our method could generate good quality mapping and state estimation. More results can be found in the supplementary video.

Quantitative results

We first evaluate on two popular sequences of TUM RGBD dataset *fr2/desk* and *fr2/xyz* shown in Fig 8.6(a). Comparison is shown in Table 8.1, where result of SDVO and two edge based VO [29] [162] are obtained from their paper. These two scenarios are feature rich environments thus are most suitable for the feature-based ORB SLAM with BA. Due to the large amounts of high gradient pixels, SDVO also performs well. Due to many curved bottles, leafs, and small keyboards, there is relatively large line detection and matching errors for these environments, our algorithm performs similarly to SDVO but better than two other edge based VO.

We also provide results on more datasets shown in Table 8.2, where other edge VO doesn't provide results. The top two scenes are relative easy environments. In TUM *mono/38* in Fig. 8.1, we only evaluate the beginning part which has ground truth pose. Since there are still some corner points on the door and showcase, ORB-SLAM with BA still performs the best but our algorithm clearly outperforms the SDVO. This is because the door surface is nearly homogeneous without

Table 8.1: Relative Position Error (cm/s) Comparison on TUM Dataset

Sequence	Ours	SDVO	ORB-SLAM	[29]	[162]
fr2/desk	1.88	2.1	**0.7**	2.8	6.9
fr2/xyz	0.66	0.6	**0.6**	0.8	2.1

Table 8.2: Relative Position Error (cm/s) Comparison on Various Datasets

Sequence	Ours	SDVO	ORB-SLAM
ICL/office2	4.44	5.72	**2.11**
mono/38	2.04	5.4	**1.16**
ICL/office1	1.85	**1.33**	X
fr3/cabinet-big	**8.82**	16.23	33.57
fr3/cabinet	**13.3**	21.7	X
fr3/notex-far	**4.32**	10	X

large intensity gradients so the photometric error minimization of SDVO doesn't work very well while our algorithm can still use edges to minimize edge re-projection error.

The last four scenes in Table 8.2 are more challenging feature-less environments shown in Fig .8.6(b) and Fig .8.6(c). ORB-SLAM doesn't work well and even fails (denoted as 'X') in some environments but direct VO can still work to some extent because direct methods utilize high gradient and edge pixels instead of feature points. Note that in *ICL/office1* dataset, the overall scene has many feature points but ORB SLAM failure happens when the camera only observes white walls and ground with few distinguishable features. Our method with line clearly outperforms SDVO in most of the cases from the table and there are mainly two reasons. Firstly, by adding edges, we are utilizing more pixels for tracking. Some pixels might have low gradients due to homogeneous surfaces but can still be utilized because of lying on edges shown in Fig .8.6(c). Secondly, we are minimizing photometric error as well as geometric error, which is known to be more robust to image noise and has a large convergence basin. This has been analysed and verified in many other works [135] [1].

To demonstrate the advantage of a large convergence basin in the optimization, we select two frames from TUM *fr3/cabinet_big* which are 41 frames apart (1.3s) and show the tracking iterations in Fig. 8.2. We can clearly see the re-projected pixels in green gradually align with the true edges in red.

8.5.3 Time analysis

We report the time usage of our algorithm running on TUM *fr3/cabinet_big* dataset shown in Table 8.3. Using two octaves of line detection, there are totally 193 edges on average per frame. The total tracking thread apart from mapping takes 51.95 ms, able to run around 20Hz. Time could vary depending on the amounts of pixels involved in the optimization. For now, edge

Table 8.3: Time analysis on TUM fr3/cabinet_big dataset.

Component	Value
Edge detection	16.24 ms
Descriptor computation	11.95 ms
Edge matching	4.52 ms
Tracking time	19.23 ms
Mapping time	10.99 ms
Edge number	193

detection and descriptor computation consumes most of the time. This could be speeded up using down-sampled image. Edline edge detector [166] can also be used to reduce detection time by half but it usually detects fewer edges compared to the currently used method [164] and may affect the state estimation accuracy in some challenging environments. Recently, Gomez *et al* [162] utilize edge tracking to decrease the computation instead of detection and matching for every frame, which is also a good solution.

8.6 Conclusions

In this paper, we propose a direct monocular odometry algorithm utilizing points and lines. We follow the pipeline of SDVO [118] and add edges to improve both tracking and mapping performance. In the tracking part, we minimize both photometric error and geometric error to the matched edges. In the mapping part, using matched edges, we can get stereo matching quickly and accurately without exhaustive search. An analytical solution is developed to regularize the depth map using edges. We also provide probability uncertainty analysis of different observation models in tracking and mapping part.

Our algorithm combines the advantage of direct and feature based VO. It is able to create a semi-dense map and the state estimation is more robust and accurate due to the incorporation of edges and geometric error minimization. On various dataset evaluation, we achieve better or comparable performance than SDVO and ORB SLAM. ORB SLAM with bundle adjustment works the best in environments with rich features. However, for scenarios with low texture, ORB SLAM might fail and direct methods usually work better. Our algorithm focuses on these scenarios and further improves the performance of SDVO by adding edges.

In the future, we want to reduce the computation of edge detection and matching by direct edge alignment. Also, bundle adjustment of edges in multiple frames could also be used to improve the accuracy. We will also exploit more information by combining points, edges, and planes [33] in one framework to improve the accuracy and robustness in challenging environments.

Chapter 9

Conclusions

9.1 Summary and Contributions

In this thesis, we have addressed the problem to jointly solve visual SLAM with semantic scene understanding. The goal is to build a semantic map at the level of cuboid object and layout planes using only monocular camera. Instead of first solving SLAM then detecting objects and planes on the point cloud, we propose to first understand the 2D image, then formulate object and plane level SLAM to improve both camera pose estimation and 3D detections. We here briefly summarize each part of the thesis.

Part I Single image understanding at Chapter 3 and 4. This part utilizes camera geometry, deep learning and graphical optimization to build a simplified 3D model composed of cuboid objects and layout planes from a single image. Compared to many existing 3D understanding work, our object detection doesn't depend on the object shape priors and the layout detection is not limited to the Manhattan box-like rooms, therefore it is more general and suitable for mobile robot navigation. To achieve that, we first propose a novel approach to generate high quality cuboid object proposals from 2D bounding box then efficiently score them based on image edge features. For the holistic understanding, we develop efficient high order graphical inference to jointly optimize the object and plane proposals based on occlusions and intersections.

Part II Tightly-coupled semantic SLAM at Chapter 5 and 6. We propose the first monocular SLAM using object and plane landmarks and provide experimental results in various indoor and outdoor, static and dynamic datasets. These two chapters are extensions to the previous part. The object and plane detections from single image understanding are directly used as SLAM landmark observations. Some novel measurement functions between cameras, cuboids and layout planes are designed. Compared to the commonly used point landmarks, these high-level landmarks can provide different semantic, geometric and long-term scale constraints to improve camera pose estimation. Meanwhile, the multi-view optimization further refines the 3D object and plane positions. Object representations are also used to improve pose estimation in dynamic environments based on the new motion constraints.

Part III Filtering based SLAM at Chapter 7 and 8. Different from the tightly-coupled

SLAM in the previous part, the part addresses two filtering based SLAM methods. One is the decoupled semantic mapping system which first builds a 3D grid map then reasons all the grid labels. The grid representation is more general compared to the structured objects and planes. The grid labels are initialized by 2D label filtering then optimized through hierarchical graph optimization. In the last chapter, we propose a direct visual odometry algorithm using another geometric representation: edge. Edge is used to combine the photometric and geometric error probabilistically and improve both tracking and mapping performance.

In all, the main research **contributions** are listed as follows:

- The first monocular visual SLAM using object and plane landmarks, demonstrating that semantic object and layout understanding and SLAM optimization can benefit each other in one system. Experiments on various indoor and outdoor datasets demonstrate the robustness and effectiveness.
- An efficient and general 3D object detections without object shape priors and layout understanding without Manhattan box room assumptions.
- The first work to utilize moving objects to improve pose estimation in dynamic scenarios.
- A real-time incremental decoupled semantic 3D grid mapping system for large-scale environments and an improved edge based direct VO.

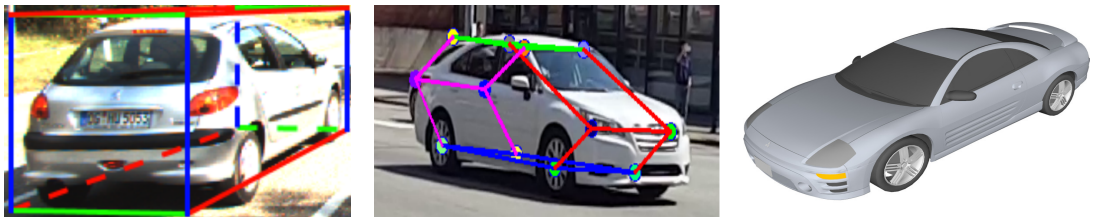
9.2 Lessons Learnt

Points Vs Objects and Planes

We have shown that object and plane landmarks can improve the state estimation performance and also benefit the dense mapping compared to point landmarks. Object representations are also more suitable for dynamic environments and the place recognition. However, there are also some limitations with them. First, there are much fewer objects and planes compared to point features, therefore it is difficult to have accurate state estimation just relying on objects if there is no strong prior information of objects. Second, objects have much larger detection error and uncertainty compared to points. Feature points only have one or two pixels location errors, but 2D object bounding box detection may have dozens of pixel localization error, which is even larger for 3D object detection. Therefore, for general feature-rich environments, it is difficult to achieve better state estimation performance using only objects. However, if there are strong shape priors for example with a complete textured 3D model, object landmarks can provide strong constraints for camera poses. The camera pose could be determined even using one image.

Tightly Vs Decoupled Semantic SLAM

One of our main contributions is a tightly coupled semantic SLAM algorithm in Chapter 6. The tight approach has the potential to improve both state estimation and semantic detections, however, it is also difficult to tune the weight parameters between different kinds of measurements such as point/object reprojection error, 3D point-object distance error, object-plane distance error etc. Compared to points, the wrong measurement or association of objects usually damage



(a) (b) (c)

Figure 9.1: Different object representation with different levels of shape priors. (a) The most general cuboid object model without prior model. (b) The object keypoint model. Different objects may have different keypoint number and structure. (c) Prior object CAD model.

the SLAM optimization more severely because there are only a few objects in the map. The bundle adjustment computation also increases because more connections between variables are built and the graph may not be sparse anymore. Therefore, in some scenarios, a decoupled approach such as Chapter 7 is preferred as it is faster and easier to implement and debug, especially when the standard SLAM state estimation is already good and object landmarks won't have large improvements on the pose estimation.

9.3 Future Work

More general environments

We have shown the object and plane detection and SLAM in various indoor and outdoor datasets and achieved improvement of accuracy and robustness compared to existing approaches. However, there are still some assumption limitations about the environment. The cuboid detection mainly applies to "boxy" objects lying on the ground. The layout understanding only applies to structured building environments and doesn't apply to the general planar surface such as tables. In the future, we are also interested in combing our cuboid object proposal generation with deep learning to automatically select the best proposal without the hand-designed edge features. This is also a widely used framework for 2D object detection. It is difficult to estimate all the surfaces' 3D locations only from a 2D image, for example, the monitor screen position cannot be reliably estimated without knowing the supporting desk's height. But for SLAM optimization, these plane surfaces can be represented similarly to the current layout planes.

With prior knowledge

In some scenarios such as autonomous driving, most cars share similar shapes. Therefore, a reasonable assumption is to use object shape priors to improve the object detection and provide stronger constraints in SLAM optimization. Apart from the general cuboid model, there are also more accurate models such as 3D keypoints [43][167] or skeletons [168] shown in Fig 9.1(b) if prior knowledge about certain object category is available. They can also be modeled and parameterized properly to have shape deformation in order to adapt to the intra-class variations. After

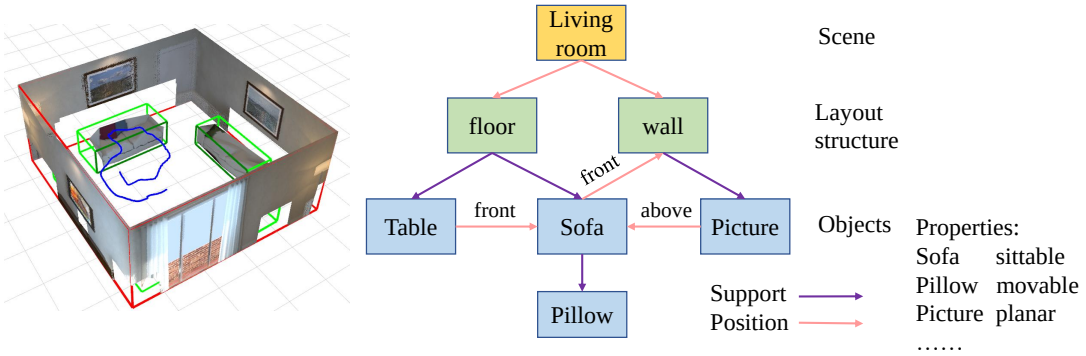


Figure 9.2: An example of scene graph. It captures more complicated supporting and positional relationships as well as different object properties. With these understanding, the robot can interact with human more intelligently.

projection, a cuboid only generates four dimensional rectangle measurement therefore loses lots of information. In contrast, there are many 2D keypoints or skeletons on object surfaces after projections, therefore, they can provide more constraints for the SLAM optimization.

Other sensors

The current system is implemented on the most general monocular camera. However, there are still many challenges when we directly apply it to real robot navigation. Monocular SLAM cannot handle scale drift robustly without the prior assumption about the environments. It is also difficult to robustly detect objects and planes using only a single image. Depth sensors such as stereo, RGBD or Laser scanner can greatly improve the perception performance and have been widely used in perception. IMU (Inertial measurement unit) is also a widely used sensor on robots to provide scale and pose constraints for state estimation. The computed roll/pitch angle can also be directly computed from IMU gravity measurement to improve our 3D ground object detection.

More than labeled map

Most existing semantic SLAM research including our work only focuses on generating a labelled map either composed of grids, objects or planes. However, apart from labels, there can be more information with a semantic map such as the property and interrelationships of concurrence, affordances, actionability and so on, shown in Fig 9.2. For instance, a sofa is usually put on planar floor aligned with the vertical wall and sit by people. Therefore, even with occlusions, detecting sofa can provide strong geometry and semantic constraints to SLAM and scene understanding. The topology of objects and layouts can also greatly benefit the long-term localization or kidnapping problem. Scene graph is recently proposed in the vision community to capture the complicated relationships between objects [169][170]. With proper parameterization and formulation, it also has the potential to be used in metric SLAM optimization.

Chapter 10

Appendix

10.1 CRF Inference for Object and Plane Understanding

We here explain the CRF inference of Section 4.3.3 in more detail. If there are N variable $x_1, x_2, ..., x_n$ in a clique $\mathbf{x}_c$. As mentioned before, there are $N+1$ special states. For each state $\mathbf{y}_k$, we define:

$$\mathbf{s}_k = \sum_{j \in \mathbf{y}_k} m_{j \to c}^{t-1}(x_j) \tag{10.1}$$

Note that all $\mathbf{s}_k$ can be computed iteratively in $O(N)$ as adjacent $\mathbf{y}_k$ is almost the same. The min and second min of $\mathbf{s}_k$ are also recorded during iterative computation. Then we can compute clique to variable i message by:

$$m_{c \to i}^{t}(x_i) = \begin{cases} \mathbf{s}_i - m_{i \to c}^{t-1}(x_i) & \text{if } x_i = 1 \\ \min\limits_{k=1:N+1, k \neq i} \mathbf{s}_k - m_{i \to c}^{t-1}(x_i) & \text{if } x_i = 0 \end{cases} \tag{10.2}$$

When $x_i = 1$, only one state $\mathbf{y}_i$ is feasible. Otherwise, we need to evaluate all $N+1$ states to find the minimum. As we already record the min and second min, evaluating Eq 10.2 only takes O(1) computation, which can also be verified from the algorithm description in Algorithm 1.

Algorithm 1: Sparse Factor-to-Variable Message Passing of Equation 4.9 and 10.2

Input: Variable-to-factor message $m_{i\to c}^{t-1}(\cdot), i = 1, ..., N$

Output: All Factor-to-variable message: $m_{c\to i}^{t}(x_i), i = 1, ..., N$

begin

  // compute all $\mathbf{s}_k$ recursively and record (second) min

  $\mathbf{s}_1 = \sum_{j=1,2,3...N} m_{j\to c}^{t-1}(\mathbf{y}_1^j)$

  $\text{minS}_{\text{val}} = \mathbf{s}_1$; $\text{sndminS}_{\text{val}} = \infty$

  for $k \leftarrow 2$ **to** $N + 1$ **do**

    if $k \leq N$ **then**

      $\mathbf{s}_k = \mathbf{s}_{k-1} - m_{k-1\to c}^{t-1}(1) + m_{k-1\to c}^{t-1}(0) - m_{k\to c}^{t-1}(0) + m_{k\to c}^{t-1}(1)$

    else

      $\mathbf{s}_k = \mathbf{s}_{k-1} - m_{k-1\to c}^{t-1}(1) + m_{k-1\to c}^{t-1}(0)$

    end

    if $\mathbf{s}_k \leq$ *minS$_{val}$* **then**

      $\text{minS}_{\text{val}} = \mathbf{s}_k$

    else

      if $\mathbf{s}_k \leq$ *sndminS$_{val}$* **then**

        $\text{sndminS}_{\text{val}} = \mathbf{s}_k$

      end

    end

  end

  // compute final message

  for $k \leftarrow 1$ **to** N **do**

    $m_{c\to k}^{t}(x_k = 1) = \mathbf{s}_k - m_{k\to c}^{t-1}(x_k = 1)$

    if $\mathbf{s}_k ==$ *minS$_{val}$* **then**

      $m_{c\to k}^{t}(x_k = 0) = \text{sndminS}_{\text{val}} - m_{k\to c}^{t-1}(x_k = 0)$

    else

      $m_{c\to k}^{t}(x_k = 0) = \text{minS}_{\text{val}} - m_{k\to c}^{t-1}(x_k = 0)$

    end

  end

end

Bibliography

[1] Raul Mur-Artal, JMM Montiel, and Juan D Tardos. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. 1.2, 2.3.1, 2.3.1, 5.1, 5.1, 5.5, 5.5.2, 6.1, 6.1, 6.2, 6.2.1, 6.3.1, 6.3.1, 6.3.2, 6.4.2, 6.5.1, 6.5.2, 6.6, 6.7, 7.1, 7.1, 7.2, 7.2.1, 8.1, 8.1, 8.5.2, 8.5.2

[2] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 1.2, 2.3.1, 6.1, 6.2.1, 6.7.1, 6.7, 6.7.3, 7.1, 8.1

[3] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. SUN RGB-D: A RGB-D scene understanding benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 567–576, 2015. 1.2, 3.3.1, 4.5.1

[4] Sid Yingze Bao, Mohit Bagra, Yu-Wei Chao, and Silvio Savarese. Semantic structure from motion with points, regions, and objects. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2703–2710. IEEE, 2012. 1.1, 2.3.3

[5] Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. SLAM+: Simultaneous localisation and mapping at the level of objects. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1352–1359, 2013. 1.1, 2.3.3, 6.1, 6.1

[6] Grace Tsai, Changhai Xu, Jingen Liu, and Benjamin Kuipers. Real-time indoor scene understanding using Bayesian filtering with motion cues. In *IEEE International Conference on Computer Vision (ICCV)*, pages 121–128. IEEE, 2011. 1.1, 3.1, 3.2.3, 5.1, 5.2.2

[7] Sudeep Pillai and John Leonard. Monocular SLAM supported object recognition. *Robotics: Science and systems*, 2015. 1.1, 2.3.3, 6.1, 6.1

[8] Dorian Gálvez-López, Marta Salas, Juan D Tardós, and JMM Montiel. Real-time monocular object SLAM. *Robotics and Autonomous Systems*, 75:435–449, 2016. 1.1, 2.3.3, 6.1, 6.1

[9] Alejo Concha, Wajahat Hussain, Luis Montano, and Javier Civera. Incorporating scene priors to dense monocular mapping. *Autonomous Robots*, 39(3):279–292, 2015. 1.1, 5.1, 5.2.2

[10] Jeong-Kyun Lee, Jaewon Yea, Min-Gyu Park, and Kuk-Jin Yoon. Joint layout estimation and global multi-view registration for indoor reconstruction. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 1.1, 2.3.3, 6.1, 6.1

[11] John McCormac, Ronald Clark, Michael Bloesch, Andrew Davison, and Stefan Leutenegger. Fusion++: Volumetric object-level SLAM. In *2018 International Conference on 3D Vision (3DV)*, pages 32–41. IEEE, 2018. 1.1, 2.3.3, 6.1

[12] Lachlan James Nicholson, Michael J Milford, and Niko Sunderhauf. QuadricSLAM: Dual quadrics from object detections as landmarks in object-oriented SLAM. *IEEE Robotics and Automation Letters*, 2018. 1.1, 2.3.3, 6.1, 6.2.2

[13] Mehdi Hosseinzadeh, Yasir Latif, Trung Pham, Niko Suenderhauf, and Ian Reid. Structure aware SLAM using quadrics and planes. *arXiv preprint arXiv:1804.09111*, 2018. 1.1, 2.3.3, 6.1, 6.5.1, 6.5.1

[14] Hauke Strasdat. *Local accuracy and global consistency for efficient visual SLAM*. PhD thesis, Citeseer, 2012. 2.1.2

[15] Jose-Luis Blanco. A tutorial on SE(3) transformation parameterizations and on-manifold optimization. *University of Malaga, Tech. Rep*, 3, 2010. 2.1.2

[16] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. iSAM: Incremental smoothing and mapping. *Robotics, IEEE Transactions on*, 24(6):1365–1378, 2008. 2.2.2, 2.3.1, 5.3.1, 5.5.4

[17] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A general framework for graph optimization. In *Robotics and Automation (ICRA), IEEE International Conference on*, pages 3607–3613. IEEE, 2011. 2.2.2, 2.3.1

[18] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007. 2.3.1

[19] Ethan Eade and Tom Drummond. Scalable monocular SLAM. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 469–476. IEEE, 2006. 2.3.1

[20] Javier Civera, Andrew J Davison, and JM Martinez Montiel. Inverse depth parametrization for monocular SLAM. *IEEE transactions on robotics*, 24(5):932–945, 2008. 2.3.1

[21] Hauke Strasdat, JMM Montiel, and Andrew J Davison. Real-time monocular SLAM: Why filter? In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2657–2664. IEEE, 2010. 2.3.1

[22] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *International Workshop on Vision Algorithms: Theory and Practice*, ICCV '99, pages 298–372, London, UK, UK, 2000. Springer-Verlag. 2.3.1

[23] Frank Dellaert and Michael Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, 25(12):1181–1203, 2006. 2.3.1

[24] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007. 2.3.1, 2.3.1, 3.1, 5.1, 8.1

[25] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*, pages 834–

849. Springer, 2014. 2.3.1, 2.3.1, 5.1, 5.1, 5.4, 5.5, 8.1, 8.1

[26] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012. 2.3.1

[27] Andreas Geiger, Julius Ziegler, and Christoph Stiller. StereoScan: Dense 3D reconstruction in real-time. In *IEEE Intelligent Vehicles Symposium*, Baden-Baden, Germany, June 2011. 2.3.1, 8.1, 8.1

[28] Ethan Eade and Tom Drummond. Edge landmarks in monocular SLAM. *Image and Vision Computing*, 27(5):588–596, 2009. 2.3.1, 8.1

[29] Juan Jose Tarrio and Sol Pedre. Realtime edge-based visual odometry for a monocular camera. In *IEEE International Conference on Computer Vision*, pages 702–710, 2015. 2.3.1, 8.1, 8.5.2, 8.5.2, 8.1

[30] Shichao Yang and Sebastian Scherer. Direct monocular odometry using points and lines. In *IEEE international conference on Robotics and automation (ICRA)*. IEEE, 2017. 2.3.1, 6.1

[31] Renato F Salas-Moreno, Ben Glocken, Paul HJ Kelly, and Andrew J Davison. Dense planar SLAM. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 157–164. IEEE, 2014. 2.3.1

[32] Michael Kaess. Simultaneous localization and mapping with infinite planes. In *International Conference on Robotics and Automation (ICRA)*, pages 4605–4611. IEEE, 2015. 2.3.1, 5.2.3, 5.3, 5.3.1, 5.3.2, 6.1, 6.2.2, 6.5.1, 6.5.1

[33] Shichao Yang, Yu Song, Michael Kaess, and Sebastian Scherer. Pop-up SLAM: a semantic monocular plane SLAM for low-texture environments. In *International conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016. 2.3.1, 4.3.1, 6.5.1, 6.6, 7.1, 8.6

[34] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. DTAM: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pages 2320–2327. IEEE, 2011. 2.3.1, 8.1, 8.3.1, 8.4.1

[35] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *IEEE International Conference on Computer Vision (ICCV)*, 2017. 2.3.2, 4.1, 6.3.2

[36] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *Computer Vision and Pattern Recognition (CVPR)*, 2017. 2.3.2, 4.4

[37] Yu Xiang, Wongun Choi, Yuanqing Lin, and Silvio Savarese. Subcategory-aware convolutional neural networks for object proposals and detection. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 924–933. IEEE, 2017. 2.3.2, 4.1, 4.5.1, 4.5.1, 4.1

[38] Florian Chabot, Mohamed Chaouch, Jaonary Rabarisoa, Céline Teulière, and Thierry Chateau. Deep MANTA: A coarse-to-fine many-task network for joint 2d and 3D vehicle analysis from monocular image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2040–2049, 2017. 2.3.2, 4.1, 4.1

[39] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond PASCAL: A benchmark for

3D object detection in the wild. In *Applications of Computer Vision (WACV), 2014 IEEE Winter Conference on*, pages 75–82. IEEE, 2014. 2.3.2

[40] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. ShapeNet: An information-rich 3D model repository. *arXiv preprint arXiv:1512.03012*, 2015. 2.3.2

[41] Shubham Tulsiani and Jitendra Malik. Viewpoints and keypoints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1510–1519, 2015. 2.3.2

[42] Abhishek Kar, Shubham Tulsiani, Joao Carreira, and Jitendra Malik. Category-specific object reconstruction from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1966–1974, 2015. 2.3.2

[43] J Krishna Murthy, GV Sai Krishna, Falak Chhaya, and K Madhava Krishna. Reconstructing vehicles from a single image: Shape priors for road scene understanding. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 724–731. IEEE, 2017. 2.3.2, 4.1, 9.3

[44] Joseph J Lim, Hamed Pirsiavash, and Antonio Torralba. Parsing IKEA objects: Fine pose estimation. In *IEEE International Conference on Computer Vision*, pages 2992–2999, 2013. 2.3.2, 4.1, 4.2.2

[45] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: Making rgb-based 3D detection and 6d pose estimation great again. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 2.3.2, 4.1

[46] Varsha Hedau, Derek Hoiem, and David Forsyth. Thinking inside the box: Using appearance models and context based on room geometry. In *European Conference on Computer Vision (ECCV)*, pages 224–237. Springer, 2010. 2.3.2, 4.1, 4.2.2

[47] Abhinav Gupta, Martial Hebert, Takeo Kanade, and David M Blei. Estimating spatial layout of rooms using volumetric reasoning about objects and surfaces. In *Advances in neural information processing systems*, pages 1288–1296, 2010. 2.3.2

[48] Alexander G Schwing, Sanja Fidler, Marc Pollefeys, and Raquel Urtasun. Box in the box: Joint 3D layout and object reasoning from single images. In *IEEE International Conference on Computer Vision*, pages 353–360, 2013. 2.3.2

[49] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3D object detection for autonomous driving. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2147–2156, 2016. 2.3.2, 4.1, 4.2.1, 4.2.1, 4.2.2, 4.5.1, 4.5.1

[50] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3D bounding box estimation using deep learning and geometry. *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, 2017. 2.3.2, 4.1, 4.2.1, 4.2.1, 4.5.1, 4.1, 6.2, 6.7.1

[51] Peiliang Li, Tong Qin, and andShaojie Shen. Stereo vision-based semantic 3D object and ego-motion tracking for autonomous driving. In *The European Conference on Computer Vision (ECCV)*, September 2018. 2.3.2, 4.1, 4.2.1, 4.2.1, 6.1

[52] Varsha Hedau, Derek Hoiem, and David Forsyth. Recovering the spatial layout of cluttered rooms. In *International Conference on Computer Vision*, pages 1849–1856. IEEE, 2009. 2.3.2, 3.1, 3.2.3, 3.4.1, 3.6, 3.4.1, 3.2, 3.3, 4.1, 5.1, 5.2.2, 5.2.3

[53] Yuzhuo Ren, Shangwen Li, Chen Chen, and C-C Jay Kuo. A coarse-to-fine indoor layout estimation (CFILE) method. In *Asian Conference on Computer Vision*, pages 36–51. Springer, 2016. 2.3.2, 4.3.1, 6.12

[54] Chuhang Zou, Alex Colburn, Qi Shan, and Derek Hoiem. LayoutNet: Reconstructing the 3D room layout from a single rgb image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2.3.2, 4.1

[55] Chen-Yu Lee, Vijay Badrinarayanan, Tomasz Malisiewicz, and Andrew Rabinovich. Roomnet: End-to-end room layout estimation. In *IEEE International Conference on Computer Vision*, 2017. 2.3.2, 4.1

[56] Dahua Lin, Sanja Fidler, and Raquel Urtasun. Holistic scene understanding for 3D object detection with RGBD cameras. In *IEEE International Conference on Computer Vision*, pages 1417–1424, 2013. 2.3.2, 4.1, 4.3, 4.3.2

[57] Andreas Geiger and Chaohui Wang. Joint 3D object and layout inference from a single RGB-D image. In *German Conference on Pattern Recognition*, pages 183–195. Springer, 2015. 2.3.2, 4.1

[58] Ankur Handa, Viorica Pătrăucean, Vijay Badrinarayanan, Simon Stent, and Roberto Cipolla. Understanding real world indoor scenes with synthetic data. 2016. 2.3.2

[59] Shubham Tulsiani, Saurabh Gupta, David Fouhey, Alexei A Efros, and Jitendra Malik. Factoring shape, pose, and layout from the 2d image of a 3d scene. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2.3.2, 4.1

[60] Sunando Sengupta, Eric Greveson, Ali Shahrokni, and Philip HS Torr. Urban 3D semantic modelling using stereo vision. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 580–585. IEEE, 2013. 2.3.3, 7.1, 7.1, 7.4.1, 7.2

[61] Alexander Hermans, Georgios Floros, and Bastian Leibe. Dense 3D semantic mapping of indoor scenes from RGB-D images. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2631–2638. IEEE, 2014. 2.3.3, 7.1, 7.1, 7.1

[62] Vibhav Vineet, Ondrej Miksik, Morten Lidegaard, Matthias Nießner, Stuart Golodetz, Victor A Prisacariu, Olaf Kähler, David W Murray, Shahram Izadi, Patrick Pérez, et al. Incremental dense semantic stereo fusion for large-scale semantic scene reconstruction. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 75–82. IEEE, 2015. 2.3.3, 7.1, 7.1, 7.1, 7.1, 7.1, 7.3.2, 7.4.1, 7.4.3, 7.2, 7.4.4

[63] Abhijit Kundu, Yin Li, Frank Dellaert, Fuxin Li, and James M Rehg. Joint semantic segmentation and 3D reconstruction from monocular video. In *European Conference on Computer Vision (ECCV)*, pages 703–718. Springer, 2014. 2.3.3, 6.1, 7.1, 7.1, 7.1, 7.4.1, 7.4.3

[64] Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3D point clouds. In *3D Vision (3DV), 2017 International*

Conference on, pages 537–547. IEEE, 2017. 2.3.3

[65] Jingming Dong, Xiaohan Fei, and Stefano Soatto. Visual-inertial-semantic scene representation for 3D object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2.3.3, 6.1

[66] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, 2010. 2.3.3, 5.1

[67] Alejo Concha and Javier Civera. DPPTAM: Dense piecewise planar tracking and mapping from a monocular sequence. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 5686–5693. IEEE, 2015. 2.3.3, 5.1

[68] Pedro Piniés, Lina Maria Paz, and Paul Newman. Dense mono reconstruction: Living with the pain of the plain plane. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5226–5231. IEEE, 2015. 2.3.3, 5.1, 8.4.1

[69] Duncan P Frost, David W Murray, et al. Object-aware bundle adjustment for correcting monocular scale drift. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4770–4776. IEEE, 2016. 2.3.3, 6.7.1, 6.7.1, 6.3

[70] Edgar Sucar and Jean-Bernard Hayet. Bayesian scale estimation for monocular SLAM based on generic object detection for correcting scale drift. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018. 2.3.3, 6.7.1, 6.7.1, 6.3

[71] Cosimo Rubino, Marco Crocco, and Alessio Del Bue. 3D object localisation from multiview image detections. *IEEE transactions on pattern analysis and machine intelligence*, 40(6):1281–1294, 2018. 2.3.3

[72] Ming Hsiao, Eric Westman, Guofeng Zhang, and Michael Kaess. Keyframe-based dense planar SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5110–5117. IEEE, 2017. 2.3.3

[73] Ashutosh Saxena, Min Sun, and Andrew Y Ng. Make3d: Learning 3D scene structure from a single still image. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):824–840, 2009. 3.1

[74] Kevin Karsch, Ce Liu, and Sing Bing Kang. Depth extraction from video using nonparametric sampling. In *Computer Vision–ECCV 2012*, pages 775–788. Springer, 2012. 3.1

[75] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *IEEE International Conference on Computer Vision*, pages 2650–2658, 2015. 3.1, 3.2.1, 3.5.2, 3.4

[76] Daniel C Lee, Martial Hebert, and Takeo Kanade. Geometric reasoning for single image structure recovery. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 2136–2143. IEEE, 2009. 3.1, 3.4.1, 3.6, 3.4.1, 3.2, 3.4.2, 3.3, 3.5.3, 5.1

[77] Alexander G Schwing, Tamir Hazan, Marc Pollefeys, and Raquel Urtasun. Efficient structured prediction for 3D indoor scene understanding. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2815–2822. IEEE, 2012. 3.1

[78] Derek Hoiem, Alexei A Efros, and Martial Hebert. Recovering surface layout from an image. *International Journal of Computer Vision*, 75(1):151–172, 2007. 3.1, 3.2.3, 3.4.1, 3.6, 3.4.1, 3.2, 3.3, 3.5.3

[79] Axel Furlan, Stephen Miller, Domenico G Sorrenti, Li Fei-Fei, and Silvio Savarese. Free your camera: 3D indoor scene understanding from arbitrary camera motion. In *British Machine Vision Conference (BMVC)*, page 9, 2013. 3.1, 3.4, 5.1

[80] Cooper Bills, Joyce Chen, and Ashutosh Saxena. Autonomous MAV flight in indoor environments using single image perspective cues. In *Robotics and automation (ICRA), IEEE international conference on*, pages 5776–5783. IEEE, 2011. 3.1

[81] Kyel Ok, Duy-Nguyen Ta, and Frank Dellaert. Vistas and wall-floor intersection features: Enabling autonomous flight in man-made environments. In *2nd Workshop on Visual Control of Mobile Robots (ViCoMoR): IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2012)*, pages 7–12, 2012. 3.1

[82] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NIPS)*, pages 1097–1105, 2012. 3.2.1, 5.1

[83] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. 3.2.1, 3.2, 3.4, 5.1, 7.3.1

[84] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected CRFs with gaussian edge potentials. *Advances in Neural Information Processing Systems (NIPS)*, 2011. 3.2.2, 3.3.3, 7.1, 7.3.2, 7.3.4, 7.4.3

[85] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Mu rphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected CRFs. *International Conference on Learning Representations (ICLR)*, 2015. 3.2.2

[86] John Edward Hershberger and Jack Snoeyink. *Speeding up the Douglas-Peucker line-simplification algorithm*. University of British Columbia, Department of Computer Science, 1992. 3.2.3

[87] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from RGBD images. In *Computer Vision–ECCV 2012*, pages 746–760. Springer, 2012. 3.3.1

[88] Jianxiong Xiao, James Hays, Krista Ehinger, Aude Oliva, Antonio Torralba, et al. SUN database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3485–3492. IEEE, 2010. 3.3.1

[89] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. LabelMe: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1-3):157–173, 2008. 3.3.1

[90] Fayao Liu, Chunhua Shen, and Guosheng Lin. Deep convolutional neural fields for depth estimation from a single image. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015. 3.3.2

[91] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *Advances in Neural Information Processing Systems (NIPS)*, pages 487–495, 2014. 3.3.2

[92] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Python for scientific computing conference (SciPy)*, volume 4, page 3. Austin, TX, 2010. 3.3.2

[93] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE conference on computer vision and pattern recognition (CVPR)*, pages 580–587, 2014. 3.5.1

[94] Jianxiong Xiao, Bryan Russell, and Antonio Torralba. Localizing 3D cuboids in single-view images. In *Advances in neural information processing systems (NIPS)*, pages 746–754, 2012. 4.1, 4.2.2, 4.5.1, 4.1

[95] Chaohui Wang, Nikos Komodakis, and Nikos Paragios. Markov random field modeling, inference & learning in computer vision & image understanding: A survey. *Computer Vision and Image Understanding*, 117(11):1610–1627, 2013. 4.1, 4.3.3

[96] Wongun Choi, Yu-Wei Chao, Caroline Pantofaru, and Silvio Savarese. Indoor scene understanding with geometric and semantic contexts. *International Journal of Computer Vision*, 112(2):204–220, 2015. 4.1

[97] Luca Del Pero, Joshua Bowdish, Daniel Fried, Bonnie Kermgard, Emily Hartley, and Kobus Barnard. Bayesian geometric modeling of indoor scenes. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2719–2726. IEEE, 2012. 4.1

[98] Hao Jiang and Jianxiong Xiao. A linear approach to matching cuboids in RGBD images. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2171–2178, 2013. 4.1

[99] Salman H Khan, Xuming He, Mohammed Bennamoun, Ferdous Sohel, and Roberto Togneri. Separating objects and clutter in indoor scenes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4603–4611, 2015. 4.1

[100] Carsten Rother, Pushmeet Kohli, Wei Feng, and Jiaya Jia. Minimizing sparse higher order energy functions of discrete variables. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1382–1389. IEEE, 2009. 4.1

[101] Nikos Komodakis and Nikos Paragios. Beyond pairwise energies: Efficient optimization for higher-order MRFs. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2985–2992. IEEE, 2009. 4.1

[102] Pushmeet Kohli and M Pawan Kumar. Energy minimization for linear envelope MRFs. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1863–1870. IEEE, 2010. 4.1

[103] Andrew Delong, Olga Veksler, Anton Osokin, and Yuri Boykov. Minimizing sparse high-order energies by submodular vertex-cover. In *Advances in Neural Information Processing*

Systems, pages 962–970, 2012. 4.1

[104] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017. 4.3.1, 6.12

[105] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009. 4.3.3

[106] Zhaowei Cai, Quanfu Fan, Rogerio Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *ECCV*, 2016. 4.4

[107] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012. 4.5.1, 6.7.1, 7.4.1

[108] Wongun Choi, Yu-Wei Chao, Caroline Pantofaru, and Silvio Savarese. Understanding indoor scenes using 3D geometric phrases. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 33–40, 2013. 4.5.1, 4.1

[109] Shichao Yang and Sebastian Scherer. CubeSLAM: Monocular 3D object detection and SLAM without prior models. *arXiv preprint arXiv:1806.00557*, 2018. 4.2, 4.5.2, 6.5.1, 6.5.1

[110] Shichao Yang, Daniel Maturana, and Sebastian Scherer. Real-time 3D scene layout from a single image using convolutional neural networks. In *IEEE international conference on Robotics and automation (ICRA)*, pages 2183 – 2189. IEEE, 2016. 5.1, 5.1, 5.2, 5.2.1, 5.2, 5.5.3, 7.3.1

[111] Alex Flint, David Murray, and Ian Reid. Manhattan scene understanding using monocular, stereo, and 3D features. In *Computer Vision (ICCV), IEEE International Conference on*, pages 2228–2235. IEEE, 2011. 5.1

[112] Rafael Grompone von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. LSD: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (4):722–732, 2008. 5.2.2

[113] Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems*, 3:19, 2012. 5.2.2, 5.2.2, 5.2.2

[114] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003. 5.2.3, 8.4.2, 8.3

[115] Carsten Rother. A new approach to vanishing point detection in architectural environments. *Image and Vision Computing*, 20(9):647–655, 2002. 5.2.3

[116] Yasuhiro Taguchi, Yong-Dian Jian, Srikumar Ramalingam, and Chen Feng. Point-plane SLAM for hand-held 3D sensors. In *Robotics and Automation (ICRA), IEEE International Conference on*, pages 5182–5189. IEEE, 2013. 5.3.2, 5.4.2

[117] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *Robotics, IEEE Transactions on*, 28(5):1188–1197, 2012. 5.3.4

[118] Jakob Engel, Jurgen Sturm, and Daniel Cremers. Semi-dense visual odometry for a

monocular camera. In *IEEE international conference on computer vision*, pages 1449–1456, 2013. 5.4.1, 8.1, 8.3.1, 8.3.2, 8.4.1, 8.4.1, 8.4.2, 8.4.3, 8.4.3, 8.5.2, 8.6

[119] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012. 5.5, 6.7.1, 6.7.1, 8.5.2

[120] Shichao Yang, Yulan Huang, and Sebastian Scherer. Semantic 3D occupancy mapping through efficient high order CRFs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017. 6.1

[121] Sid Yingze Bao, Axel Furlan, Li Fei-Fei, and Silvio Savarese. Understanding the 3D layout of a cluttered room from multiple images. In *IEEE Winter Conference on Applications of Computer Vision*, pages 690–697. IEEE, 2014. 6.1

[122] Jianxiong Xiao and Yasutaka Furukawa. Reconstructing the worlds museums. *International journal of computer vision*, 110(3):243–258, 2014. 6.1

[123] Ioan Andrei Bârsan, Peidong Liu, Marc Pollefeys, and Andreas Geiger. Robust dense mapping for large-scale dynamic environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018. 6.1

[124] Berta Bescós, José M Fácil, Javier Civera, and José Neira. DynSLAM: Tracking, mapping and inpainting in dynamic scenes. *IEEE Robotics and Automation Letters*, 2018. 6.1

[125] Shiyu Song and Manmohan Chandraker. Joint SFM and detection cues for monocular 3D localization in road scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3734–3742, 2015. 6.1, 6.7.2, 6.5

[126] N Dinesh Reddy, Prateek Singhal, Visesh Chari, and K Madhava Krishna. Dynamic body VSLAM with semantic constraints. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 1897–1904. IEEE, 2015. 6.1

[127] Mina Henein, Gerard Kennedy, Robert Mahony, and Viorela Ila. Exploiting rigid body motion for SLAM in dynamic environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018. 6.1

[128] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006. 6.4.1

[129] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015. 6.4.2

[130] A. Handa, T. Whelan, J.B. McDonald, and A.J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *IEEE Intl. Conf. on Robotics and Automation, ICRA*, Hong Kong, China, May 2014. 6.7.1, 6.7.3, 8.5.2

[131] Bhoram Lee, Kostas Daniilidis, and Daniel D Lee. Online self-supervised monocular visual odometry for ground vehicles. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 5232–5238. IEEE, 2015. 6.7.1, 6.3

[132] Shiyu Song, Manmohan Chandraker, and Clark C Guest. High accuracy monocular SFM and scale correction for autonomous driving. *IEEE transactions on pattern analysis and*

machine intelligence, 38(4):730–743, 2016. 6.7.1, 6.3

[133] Shiyu Song and Manmohan Chandraker. Robust scale estimation in real-time monocular SFM for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1566–1573, 2014. 6.5

[134] Yan Lu and Dezhen Song. Robustness to lighting variations: An RGB-D indoor visual odometry using line segments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 688–694. IEEE, 2015. 6.7.3

[135] J. Engel, V. Usenko, and D. Cremers. A photometrically calibrated benchmark for monocular visual odometry. In *arXiv:1607.02555*, July 2016. 6.7.3, 8.5.2, 8.5.2, 8.5.2

[136] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. ENet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016. 6.7.4, 7.4.4

[137] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011. 7.1

[138] Zheng Fang, Shichao Yang, Sezal Jain, Geetesh Dubey, Stephan Roth, Silvio Maeta, Stephen Nuske, Yu Zhang, and Sebastian Scherer. Robust autonomous flight in constrained and visually degraded shipboard environments. *Journal of Field Robotics*, 34(1):25–52, 2017. 7.1, 7.2, 7.2.1, 8.1

[139] Sebastian Scherer, Joern Rehder, Supreeth Achar, Hugh Cover, Andrew Chambers, Stephen Nuske, and Sanjiv Singh. River mapping from a flying robot: state estimation, river detection, and obstacle mapping. *Autonomous Robots*, 33(1-2):189–214, 2012. 7.1, 7.2

[140] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013. 7.1

[141] Sunando Sengupta and Paul Sturgess. Semantic octree: Unifying recognition, reconstruction and representation via an octree constrained higher order MRF. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1874–1879. IEEE, 2015. 7.1, 7.1, 7.4.1, 7.4.3, 7.2

[142] Zhe Zhao and Xiaoping Chen. Building 3D semantic maps for mobile robots using RGB-D camera. *Intelligent Service Robotics*, 9(4):297–309, 2016. 7.1, 7.1, 7.1

[143] Xuanpeng Li and Rachid Belaroussi. Semi-dense 3D semantic mapping from monocular SLAM. *arXiv preprint arXiv:1611.04144*, 2016. 7.1

[144] Vibhav Vineet, Jonathan Warrell, and Philip HS Torr. Filter-based mean-field inference for random fields with higher-order terms and product label-spaces. *International Journal of Computer Vision*, 110(3):290–307, 2014. 7.1, 7.3.4, 7.4.3

[145] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions.

International Conference on Learning Representations (ICLR), 2016. 7.2, 7.3.1, 7.4.1

[146] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Susstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(11):2274–2282, 2012. 7.2

[147] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In *Asian conference on computer vision*, pages 25–38. Springer, 2010. 7.2.1

[148] Jamie Shotton, Matthew Johnson, and Roberto Cipolla. Semantic texton forests for image categorization and segmentation. In *Computer vision and pattern recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 7.3.1, 7.2, 7.4.3

[149] Pushmeet Kohli, M Pawan Kumar, and Philip HS Torr. P3 & beyond: Solving energies with higher order cliques. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007. 7.3.3, 7.3

[150] Pushmeet Kohli, Philip HS Torr, et al. Robust higher order potentials for enforcing label consistency. *International Journal of Computer Vision*, 82(3):302–324, 2009. 7.3.3

[151] Chris Russell, Pushmeet Kohli, Philip HS Torr, et al. Associative hierarchical CRFs for object class image segmentation. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 739–746. IEEE, 2009. 7.3.3

[152] Daniel Maturana. Scrollgrid, http://dx.doi.org/10.5281/zenodo.832977, August 2016. 7.4.1

[153] Julien PC Valentin, Sunando Sengupta, Jonathan Warrell, Ali Shahrokni, and Philip HS Torr. Mesh based semantic modelling for indoor and outdoor scenes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2067–2074, 2013. 7.2, 7.4.3

[154] Albert S Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual odometry and mapping for autonomous flight using an RGB-D camera. In *International Symposium on Robotics Research (ISRR)*, volume 2, 2011. 8.1

[155] Rubén Gómez-Ojeda and Javier González-Jiménez. Robust stereo visual odometry through a probabilistic combination of points and line segments. 2016. 8.1, 8.1

[156] Manohar Prakash Kuse and Shaojie Shen. Robust camera motion estimation using direct edge alignment and sub-gradient method. In *IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden*, 2016. 8.1, 8.1

[157] Raúl Mur-Artal and Juan D Tardós. Probabilistic semi-dense mapping from highly accurate feature-based monocular SLAM. *Proceedings of Robotics: Science and Systems, Rome, Italy*, 2015. 8.1

[158] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Dense visual SLAM for RGB-D cameras. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2100–2106. IEEE, 2013. 8.1

[159] Georg Klein and David Murray. Improving the agility of keyframe-based SLAM. In *European Conference on Computer Vision*, pages 802–815. Springer, 2008. 8.1

[160] Jonas Witt and Uwe Weltin. Robust stereo visual odometry using iterative closest multiple lines. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4164–4171. IEEE, 2013. 8.1

[161] Yan Lu and Dezhen Song. Robust RGB-D odometry using point and line features. In *IEEE International Conference on Computer Vision*, pages 3934–3942, 2015. 8.1, 8.4.4, 8.4.4, 8.5

[162] Ruben Gomez-Ojeda, Jesus Briales, and Javier Gonzalez-Jimenez. PL-SVO: Semi-direct monocular visual odometry by combining points and line segments. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4211–4216. IEEE, 2016. 8.1, 8.5.2, 8.5.2, 8.1, 8.5.3

[163] Richard M Murray, Zexiang Li, S Shankar Sastry, and S Shankara Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 1994. 8.2.2

[164] von Gioi R Grompone, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. LSD: a fast line segment detector with a false detection control. *IEEE transactions on pattern analysis and machine intelligence*, 32(4):722–732, 2010. 8.5.1, 8.5.3

[165] Lilian Zhang and Reinhard Koch. An efficient and robust line segment matching approach based on lbd descriptor and pairwise geometric consistency. *Journal of Visual Communication and Image Representation*, 24(7):794–805, 2013. 8.5.1

[166] Cuneyt Akinlar and Cihan Topal. EDLines: A real-time line segment detector with a false detection control. *Pattern Recognition Letters*, 32(13):1633–1642, 2011. 8.5.3

[167] N Dinesh Reddy, Minh Vo, and Srinivasa G Narasimhan. CarFusion: Combining point tracking and part detection for dynamic 3D reconstruction of vehicles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1906–1915, 2018. 9.3

[168] Jiajun Wu, Tianfan Xue, Joseph J Lim, Yuandong Tian, Joshua B Tenenbaum, Antonio Torralba, and William T Freeman. Single image 3D interpreter network. In *European Conference on Computer Vision*, pages 365–382. Springer, 2016. 9.3

[169] Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, 2017. 9.3

[170] Michael Ying Yang, Wentong Liao, Hanno Ackermann, and Bodo Rosenhahn. On support relations and semantic scene graphs. *ISPRS journal of photogrammetry and remote sensing*, 131:15–25, 2017. 9.3