POLITECNICO DI TORINO

CORSO DI LAUREA MAGISTRALE in MECHATRONIC ENGINEERING

POLITECNICO
DI TORINO

3D reconstruction of real environment from images taken from UAV (SLAM approach)

Tesi di:
Rentao Sun
matricola 233318

Relatore:
Prof. BOTTINO ANDREA GIUSEPPE (DAUIN)

Anno Accademico 2017-2018

Contents

List of Figures

List of Tables

I would like to thank to Prof. BOTTINO ANDREA GIUSEPPE for all of his patient and guidance through this process, all the discussion, ideas, and feedback have been absolutely invaluable. I am very grateful to you.
Especially, I would like to thank my friends and parents for their continuous support and encouragement in the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Chapter 1

Introduction

1.1 Motivation

With the decreasing of the drone cost, it will be used more common in construction industry. This thesis is to find a way that make a reconstruction task run a UAV be possible only based on a monocular camera. The goal that this method need to be easy to implementation and fast to get a result. So the SLAM approach is a suitable method to be considered. We will find a suitable solution based on exist SLAM algorithms.

1.2 Background

An unmanned aerial vehicle (UAV), commonly known as a drone, is an aircraft without a human pilot aboard. With the development of this industrial, cost of a drone is becoming more economic and acceptable, some operations with too much cost have been practical in now days. Such as land surveying. For example, DJI Enterprise gives an integrity drone product for media and also land surveying. Amazon

also develops a drone system called Prime Air, to uses drone to offer a rapid logistics. This will send the drone to more complex environment and condition.

There is a problem called SLAM, "simultaneous localization and mapping (SLAM)" 1.1, which is constructing a map of an unknown environment while keeping tack of the location. This can make the device such as drone to build the environment and know the relative relation between itself and the environment to avoid collision, this is a fundamental to develop a automatic drone drive system. The basic concept of SLAM is developed at about 1990s, but until 2000s, with the competition of the DARPA Grand Challenge, an executable solution just appear, and with the development of commercial self-driving cars, the SLAM technology is also developing so quickly. To benefit from the SLAM technology development in self-driving area, the SLAM for the drone can be also concerned. But different from the competition, in daily life, the cost of a device is an important term to be considered, so the device to operate should be economic and robust.

To interact with the environment, we need to use different sensor, some sensor is robust and precise, such as Lidar and GPS ,but also costly, and some others may be opposite, such as a camera. A SLAM solution based on the camera is called "visual SLAM". Visual SLAM has some advantages, such as ability to recognize the texture of object, this means, when you drive or fly you can know there is a hard obstacle or just a cloud, and could even predict this object next move. Also, there are some disadvantages of visual SLAM, such as work poorly under undesired light condition and much noise in the output map. So, in common, there is a fusion between Lidar, GPS, and camera to do the SLAM to make sure the result be best and robust. But this will lead the increasing of the cost of the whole device. And also a big

problem in the drone SLAM, the physical weight and size of a sensor will be more considered due to the limitation of the lift force. So the camera will be suitable solution in some cases.

There are three main kinds of camera we can get, stereo camera 1.2, RGB-D camera 1.3 where D means depth, and only a single optical camera also called monocular camera. In this thesis, we will work with monocular camera. One of the main reasons to use only one camera is that the hardware is much simpler, this means it is more economic and physically smaller than stereo SLAM and even Lidar. So monocular SLAM should be most general and various in daily life.

1.3 Implementation

There has been some developed algorithms in monocular SLAM field. In this thesis, we will test two main algorithms which are state-of-the-art in this field: ORB-SLAM and LSD-SLAM as shown in Chapter 2. The test is hold on the ROS (Robot Operating System) platform, the ROS version is jade and run on Ubuntu 16.04 , this because there are many developed package about computer vision on ROS, and it is compatible with PC. And the video resource is all from Youtube to demonstrate that this solution can be taken in various environments. And use a ROS package "video-stream-opencv" that could publish the image from a video file to a ros topic, this could allow us to test customizable video from different source, not only limit in the ros package file. After we found that the LSD-SLAM is more suitable than ORB-SLAM to operate a reconstruction task. So we select LSD-SLAM to be a candidate method (as in Chapter 2).

When we test more on LSD-SLAM, we also find there exist some problem with this algorithm, especially under rotation condition. So

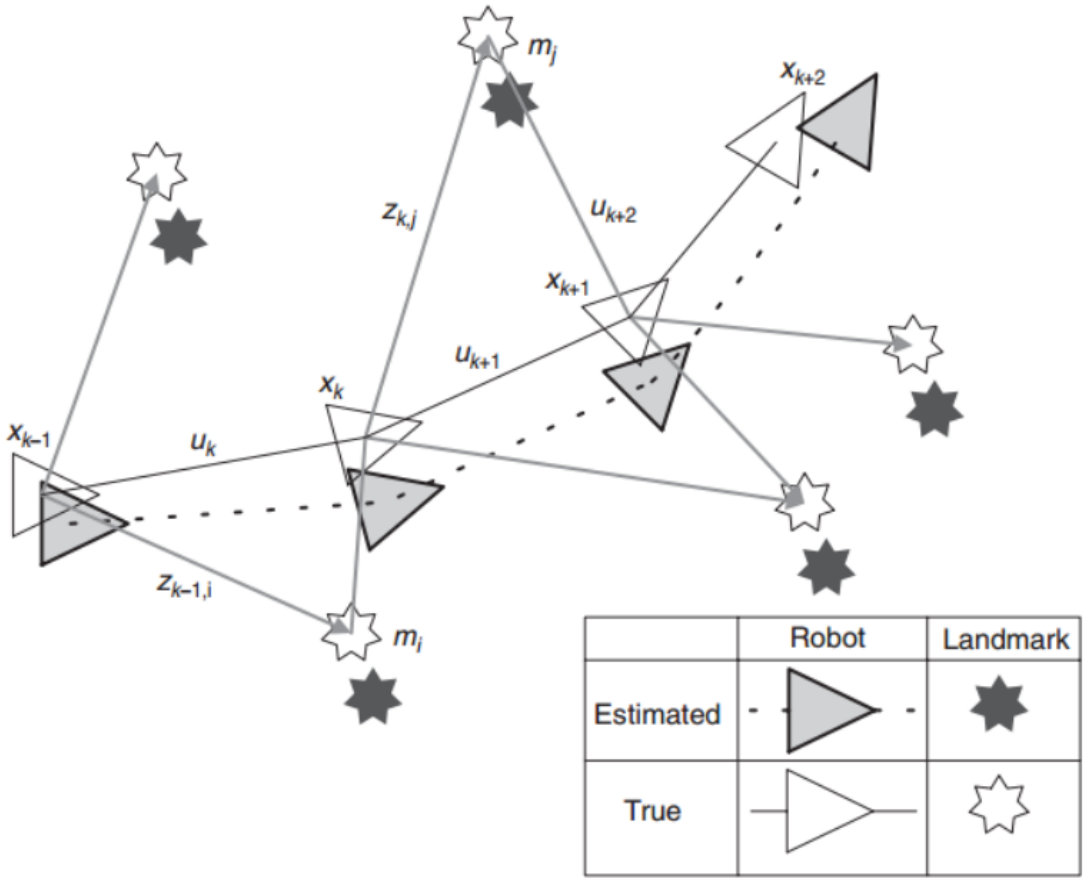

Figure 1.1: SLAM

refer to a modern device Kinect camera which is a RGB-D camera, this camera could get a better result than monocular camera, we decide to add depth information to the LSD-SLAM, but not the same as ORB-SLAM, there is no RGB-D mode in LSD-SLAM, so we need to understand the main process of it and disassemble the code, to know where to insert the depth data (Chapter 3).

Benefit from much researching on deep learning field, there is so much achievement on the depth estimation from a single image. So we need to choose a suitable way to provide a depth data. Different from

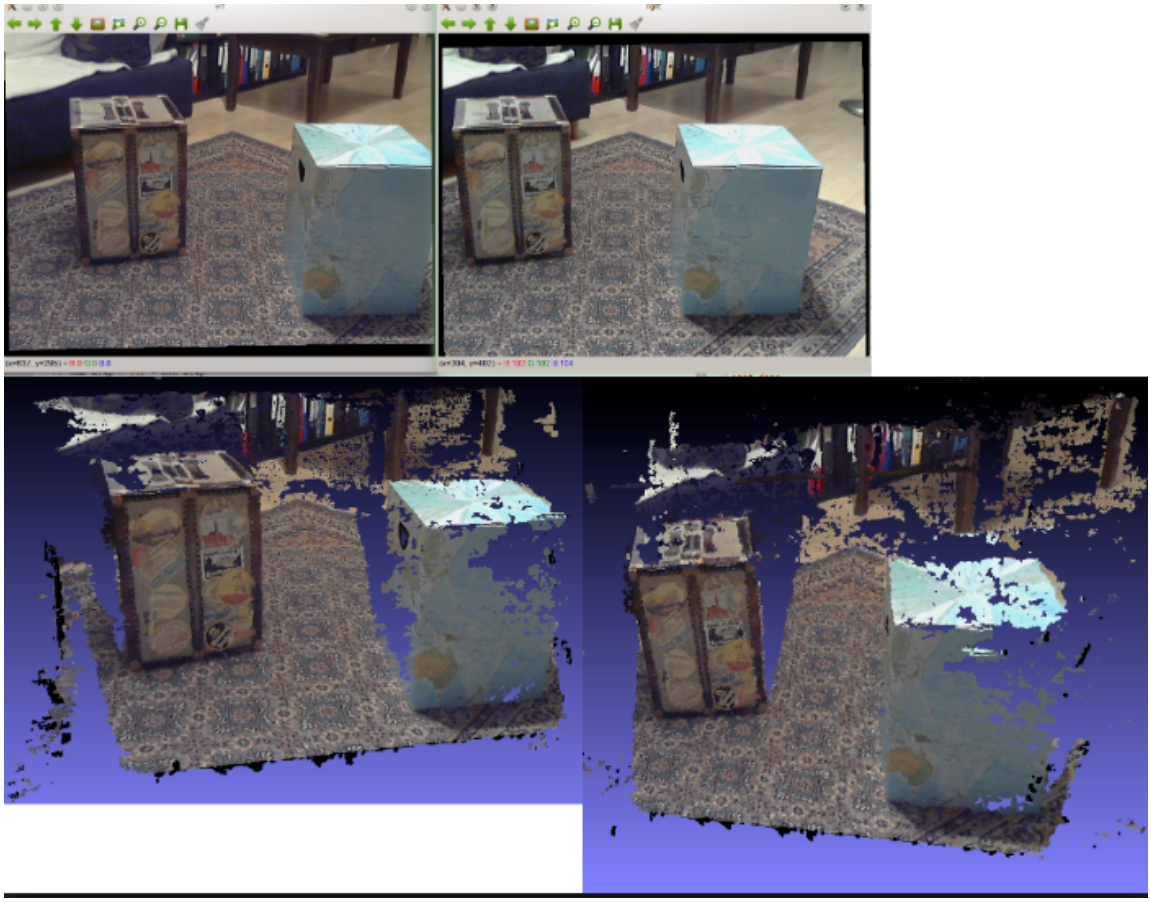

Figure 1.2: stereo camera

the indoor experiment or self-driving car. The external environment of a drone is that, the object is much far away from the sensor and the noise is much more. According to most approach is that train the network based on the image and the ground truth from the Lidar sensor, this kind of approach is not suitable to train a network correspond the image from the drone, because that the ground truth is hard to get from a long distance which is the most usual condition in a drone. So we select that a CNN depth estimation based on the left-right image consistency (Chapter 4), that trains a network only based on input from stereo camera, and try to make the a intermediate result from a

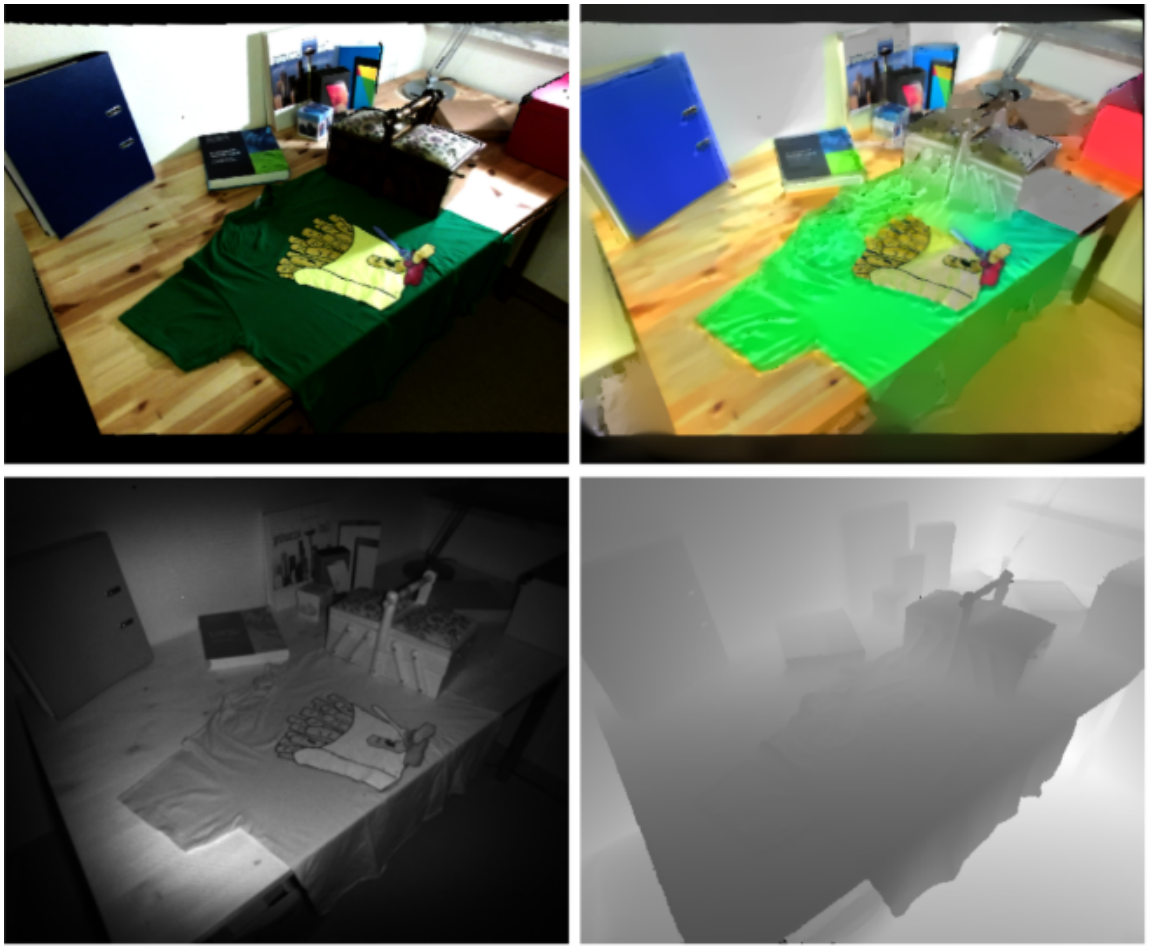

Figure 1.3: RGB-D camera

side corresponds to input image from opposite side. So that we can treat this as stereo images and calculate the disparity to estimate the depth data. Because that, as a training set data, the image from a stereo camera is much easier to achieve than the ground truth data from the Lidar sensor on a drone.

At last (Chapter 5), we test the LSD-SLAM under a strict trajectory control, after a automatic denoise and build a raster model, then compare to the ground truth. We get that the LSD-SLAM is a suitable method to this problem. Then we verify that the improved

LSD-SLAM with CNN depth estimation as mentioned in Chapter 4, under rotation condition, we can get that the CNN depth estimation could give a improvement to LSD-SLAM.

Chapter 2

State of the art monocular SLAM algorithms

2.1 Introduction to SLAM

First, I would like to introduce a kind of computational problem "simultaneous localization and mapping(SLAM)", which is constructing a map of an unknown environment while keeping tack of the location. This problem would be a chicken-and-egg problem, since a device needs to localize itself according to the environment, but the environment is unknown, so it needs also to build the map of environment respect to itself, while both key elements are unknown. To solve this problem, some probabilistic approach is employed, such as extended Kalman filter and particle filters (Monte Carlo methods). To develop the robotic technical, this is a fundamental problem to solve. So many people and agents are investigating in this field. This can make the device

such as drone to build the environment and know the relative relation between itself and the environment to avoid collision, also, in drone industrial, this is a fundamental to develop a automatic drone drive system. According to interact with the environment, we need to use different sensor, some sensor is robust and precise, such as Lidar and GPS ,but also costly, and some others may be opposite, such as a camera. Works with only one camera is called"monocular SLAM", one of the main reasons to use only one camera is that the hardware is much simpler, this means it is more economic and physically smaller than stereo SLAM and even Lidar. So monocular SLAM should be most general and various in daily life.

In this field, many different algorithms have been implemented. They can be categorized in two main types, direct and feature based.

Feature based method process the input image to extract some feature firstly, especially corner, and match features, usually Random sample consensus (RANSAC), then minimize reprojection error to process. This method can be more accuracy, due to efficient optimization of structure and motion (Bundle Adjustment). But slow due to costly feature extraction and matching. Also it perform inefficiency in some conditions that lack of corners 2.2.

Direct method compare the entire images to each other to minimize photometric error, it can use all information in the image, so it can work in some environment with less corner such as mostly edges 2.3.

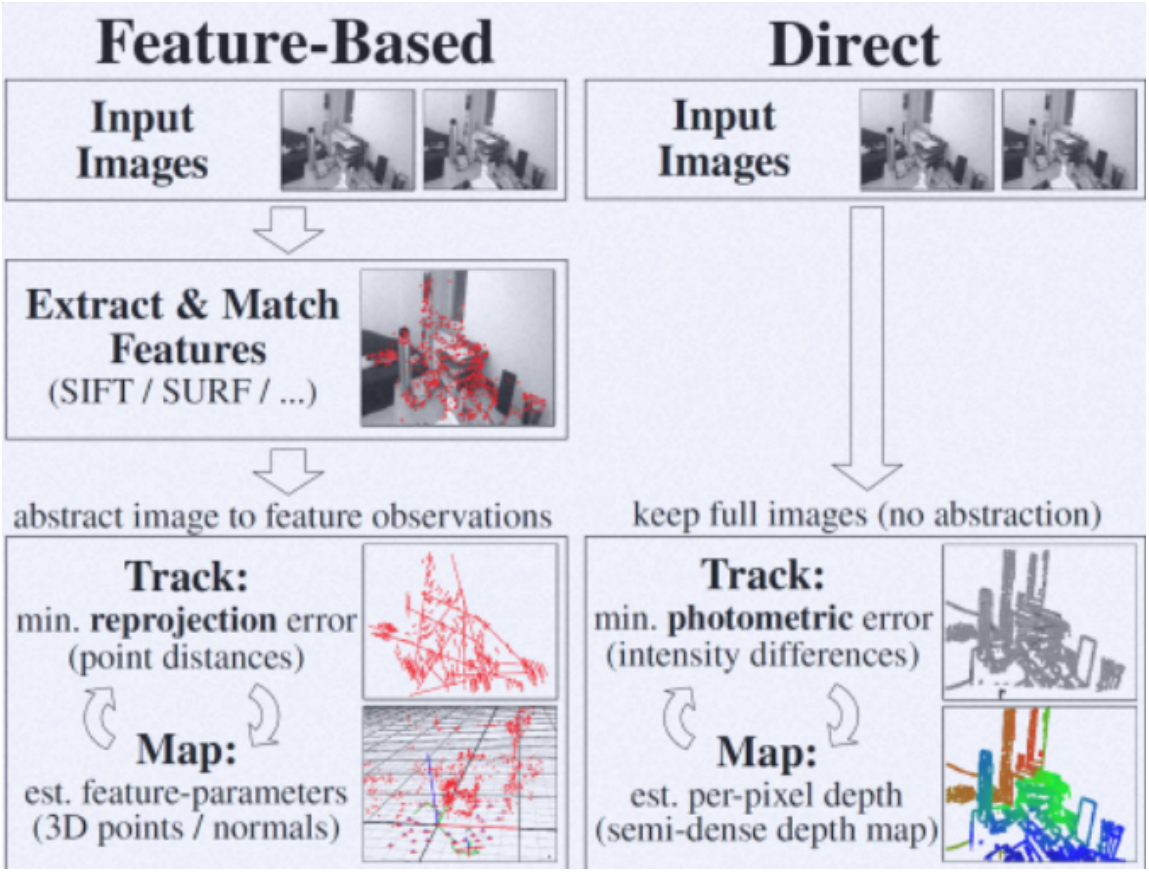

Figure 2.1: main steps of direct and feature based SLAM

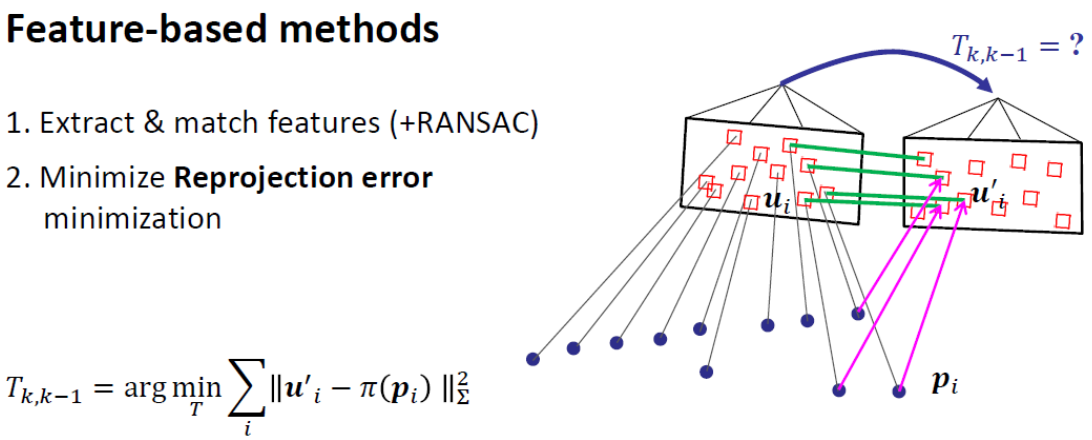

Figure 2.2: feature-based

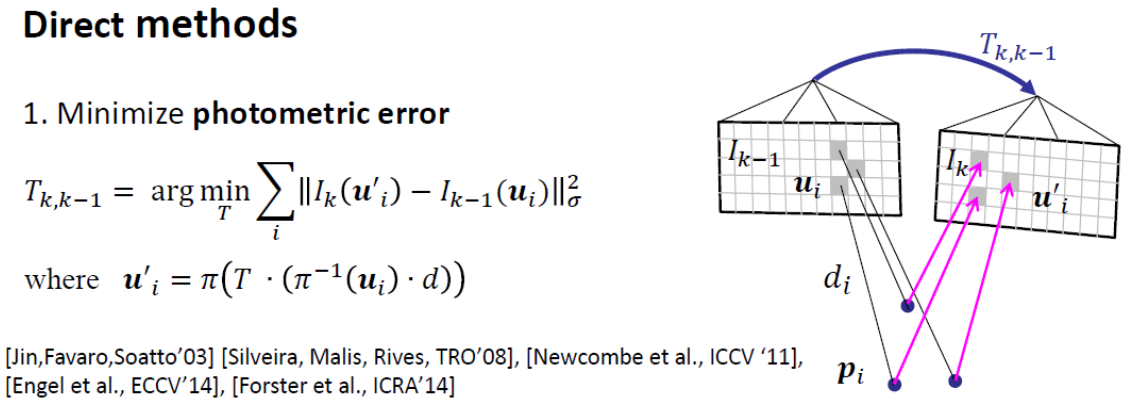

Figure 2.3: direct method

2.2 Classic SLAM algorithms

The SLAM problem can be defined as: with a sequence of observations u_t, z_t in discrete time t, to compute the most possible location x_t and the map of environment m_t, so we can write it as :

$$P(m_t, x_t | u_{1:t}, z_{1:t}) \tag{2.1}$$

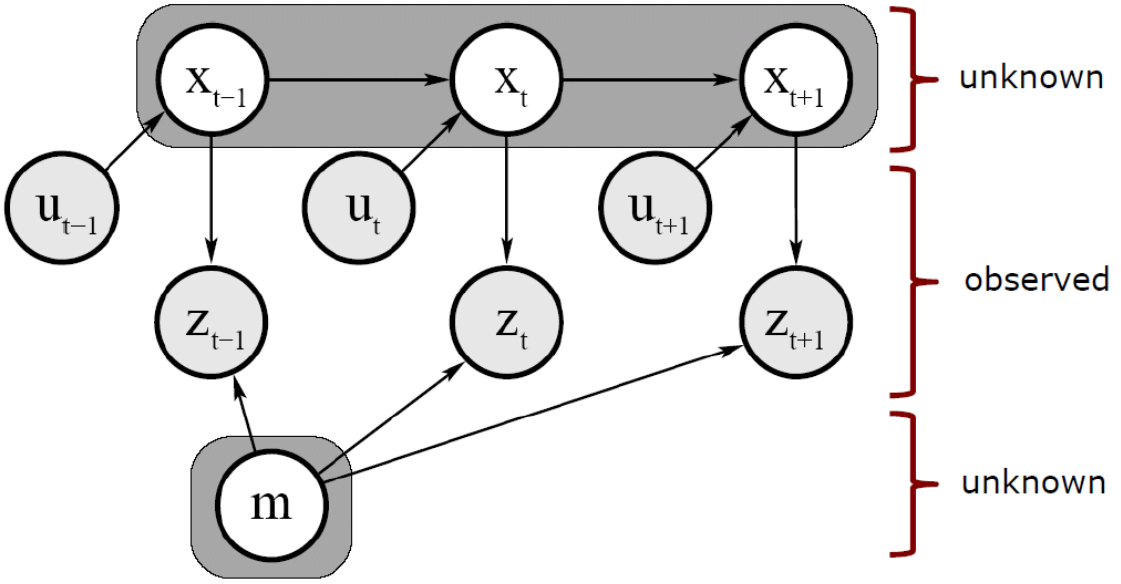

Figure 2.4: slam graphical model

In DARPA Grand Challenge, Sebastian Thrun with Stanley team first shows a great implementation of a SLAM algorithm, FastSLAM which uses particle filter. The key step of FastSLAM is that compare the every estimation of landmarks with the every estimation of position.

- Extend the path posterior by sampling a new pose for each sample

$$x_t^{[k]} \sim p(x_t | x_{t-1}^{[k]}, u_t) \tag{2.2}$$

- Computer particle weight

$$w^{[k]} = |2\pi Q|^{1/2} exp\{-1/2(z_t - \hat{z}^{[k]})^T Q^{-1}(z_t - \hat{z}^{[k]})\} \tag{2.3}$$

- Update belief of observed landmarks (EKF update rule)

- Resample

Where Q is measurement covariance and $\hat{z}$ is exp. observation.

So in general, the SLAM can be represented as

- Feature detection
- Pose estimation
- Pose graph optimization (EKF, particle filter)
- Loop closure

After that, we can get a output of robot path in a environment map. This map is what we want in reconstruction task.

2.3 State of the art monocular SLAM

In feature based SLAM, the ORB-SLAM 2.5ssss developed by the Zaragoza University is the state of the art now. It use the oriented FAST and rotated BRIEF feature to match, then the bundle adjustment to optimize.

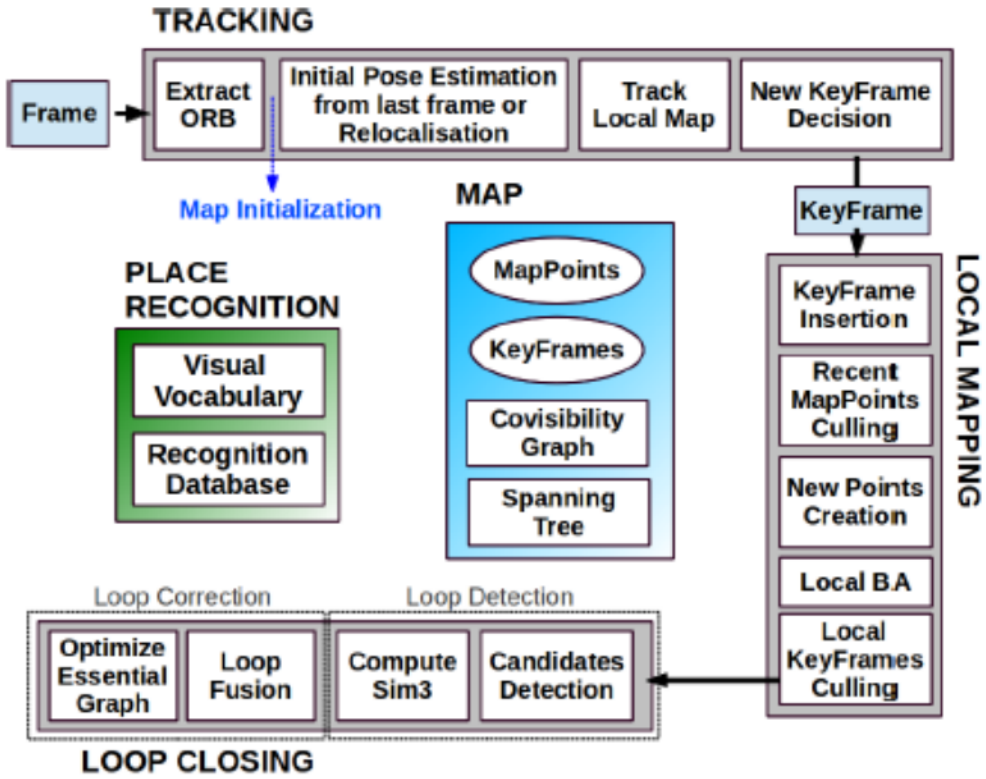

Figure 2.5: ORB-SLAM workflow

In direct SLAM, the LSD-SLAM develop by Technical University of Munich is the state of art now. In this thesis, the result is based on this algorithm and there is a improvement on it.

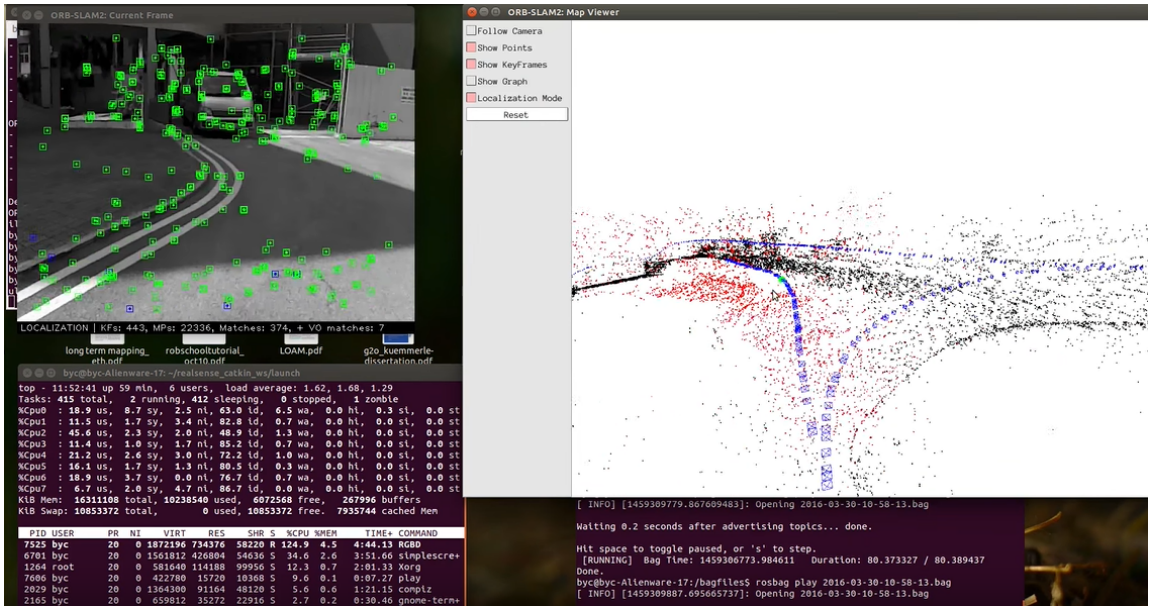

Figure 2.6: ORB-SLAM

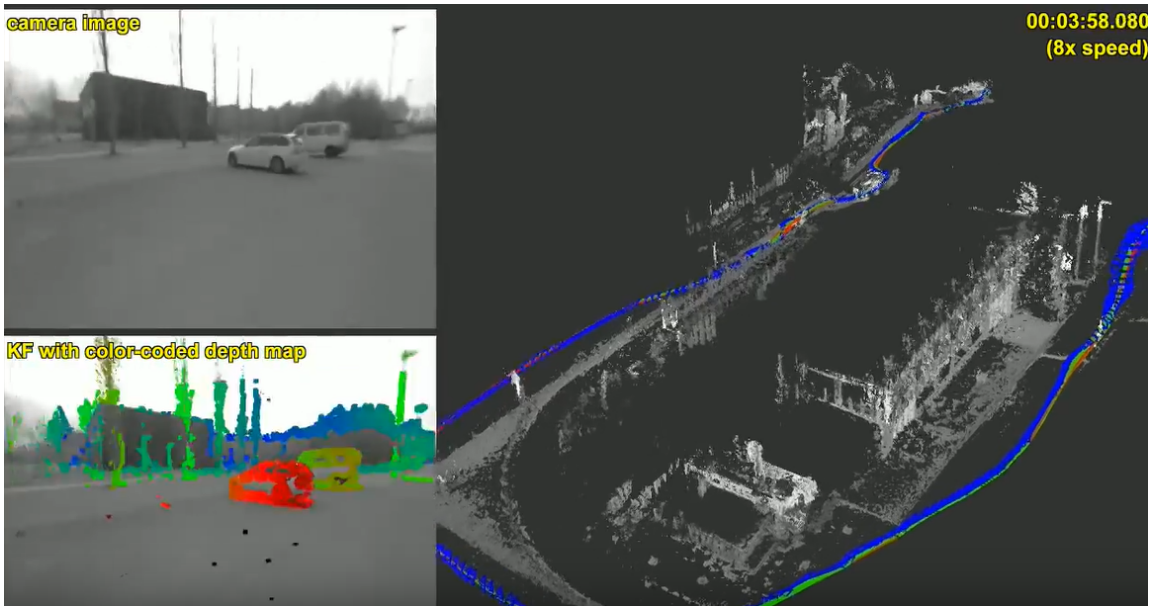

Figure 2.7: LSD-SLAM

As we can see in 2.6 and 2.7, the feature-based method only generate a sparse point cloud, this method is good at location part in the

SLAM problem. And the direct method can generate a semi-dense point cloud, this means this method is better in mapping part of the SLAM problem. So compare in these two state of the art algorithms, we can say that direct method is better to do a reconstruction task. Meanwhile LSD-SLAM is chosen.

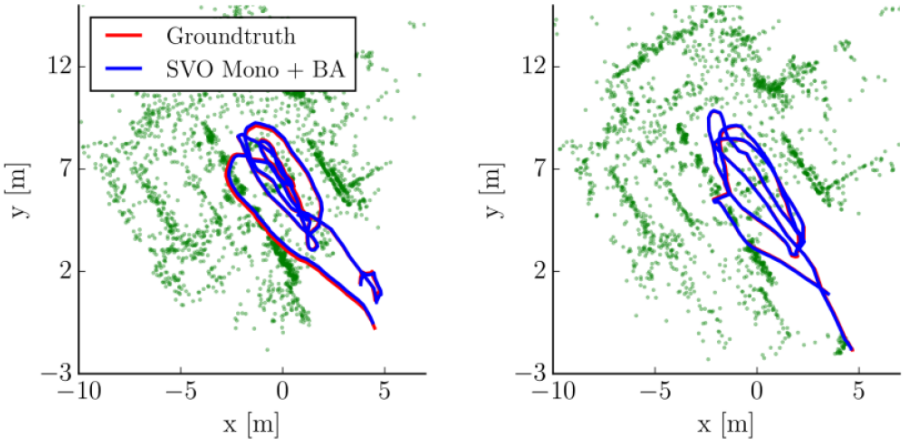

Figure 2.8: path to test LSD and ORB

	Euroc 1 RMS Error	Euroc 2 RMS Error	Timing
ORB SLAM	0.11 m	0.19 m	29.81 ms
LSD SLAM	0.13 m	0.43 m	23.23 ms

Table 2.1: result of test

Chapter 3

LSD-SLAM

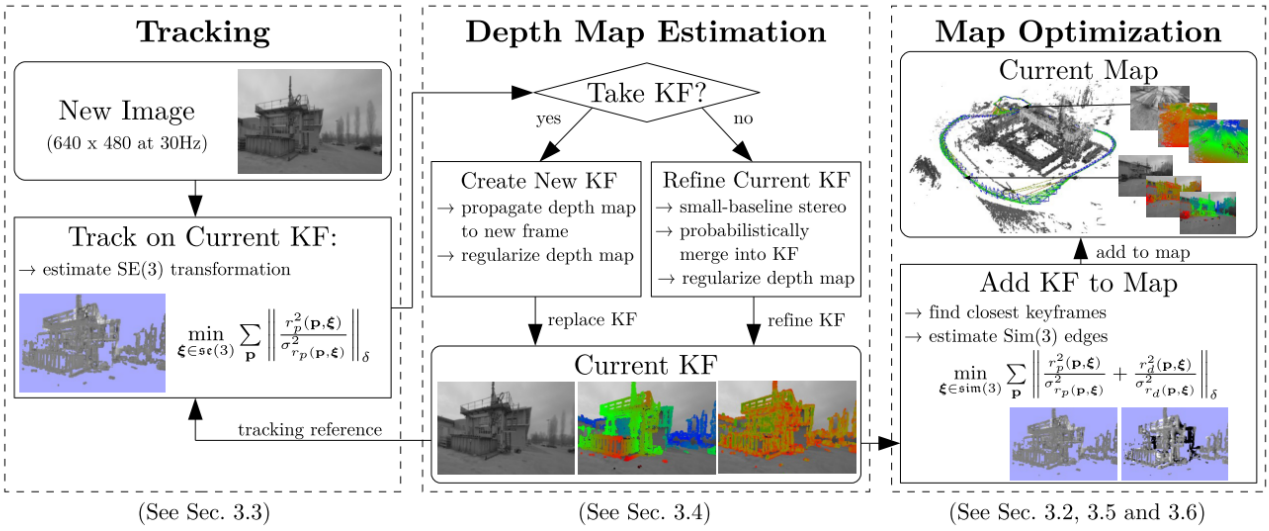

Figure 3.1: Overview over the complete LSD-SLAM algorithm

The algorithm can be divided into three main parts.

- Tracking
- Depth map estimation
- Map optimization

3.1 Tracking

In direct method, relative 3D pose $\xi_{ji} \in \mathrm{SE}(3)$ of a new input image I_j is calculated by minimizing the photometric error, where SE(3) is denoted as rotation and translation in 3D.

$$E_p(\xi_{ji}) = \sum_{\mathbf{p}\in\Omega_{D_i}} \left\| \frac{r_p^2(\mathbf{p}, \xi_{ji})}{\sigma^2_{r_p(\mathbf{p},\xi_{ji})}} \right\|_\delta \tag{1}$$

$$r_p(\mathbf{p}, \xi_{ji}) := I_i(\mathbf{p}) - I_j(\omega(\mathbf{p}, D_i(\mathbf{p}), \xi_{ji})) \tag{2}$$

$$\sigma^2_{r_p(\mathbf{p},\xi_{ji})} := 2\sigma_I^2 + (\frac{\partial r_p(\mathbf{p}, \xi_{ji})}{\partial D_i(\mathbf{p})})^2 V_i(\mathbf{p}) \tag{3}$$

$\| * \|_\delta$ is Huber-norm, computed as

$$\|r^2\|_\delta := \begin{cases} \frac{r^2}{2\delta} & \text{if} \quad |r| \le \delta \\ |r| - \frac{\delta}{2} & \text{otherwise} \end{cases} \tag{4}$$

To minimize the $E_p(\xi_{ji})$, we can get the input image pose respect to the keyframe. Meanwhile, we suppose there is a Gaussian image intensity noise σ_I^2, Minimization is operated using interatively re-weighted Gauss-Newton optimization.

3.1.1 Weighted Gauss-Newton Optimization on Lie-Manifolds

Two images are processed by Gauss-Newton minimization of the photometric error

$$E(\xi) = \sum_i \underbrace{\left(I_{ref}(\mathbf{p}_i) - I(\omega(\mathbf{p}, D_{ref}(\mathbf{p}), \xi))\right)^2}_{=:r_i^2(\xi)} \tag{5}$$

this provide the maximum-likelihood estimator for assuming independent and identically distributed Gaussian residuals. To perform a left-compositional formulation, start with an initial condition $\xi^{(0)}$,in every iteration, an increment $\xi^{(n)}$ considered . We convert equation (5) to

$$\delta\xi^* = \arg\min_{\delta\xi} E(\delta\xi \circ \xi) = \arg\min_{\delta\xi} \sum_i r_i^2(\delta\xi \circ \xi) \tag{5*}$$

Then to computer Taylor serious of the r_i:

$$r_i(\delta\xi \circ \xi) = r_i(\xi) + \mathbf{J}_i\delta\xi \tag{3.1}$$

Then put this equation into equation (5*), then calculate the derivative, and make the derivative equal to zero. We can get the increment as

$$\delta\xi^{(n)} = -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{r}(\xi^{(n)}) \quad \text{with} \quad \mathbf{J} = \left.\frac{\partial\mathbf{r}(\epsilon \circ \xi^{(n)})}{\partial\epsilon}\right|_{\epsilon=0} \tag{6}$$

While $\mathbf{J}$ is the derivative of the residual vector $\mathbf{r} = (r_1, \ldots, r_k)^T$ and r_i is the $r_i(\xi)$ in equation (5), and $\mathbf{J}^T\mathbf{J}$ is the Gauss-Newton approximation of the Hessian of E. Then we get a new estimate

$$\xi^{(n+1)} = \delta\xi^{(n)} \circ \xi^{(n)} \tag{7}$$

To reduce negative effect of outliers, different weighting-schemes is added, and convert into an interactively re-weighted least-square problem: in every times of computation, a weight matrix $\mathbf{W} = \mathbf{W}(\xi^{(n)})$ is added to down-weights large residuals, the solved error function

becomes:

$$E(\xi) = \sum_i \omega_i(\xi) r_i^2(\xi) \tag{8}$$

and the updated is :

$$\delta\xi^{(n)} = -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{W}\mathbf{r}(\xi^{(n)}) \tag{9}$$

While implantation the weight is Huber-weight

$$w(r)_\delta := \begin{cases} 1 & \text{if} \quad |r| \leq \delta \\ \\ \frac{\delta}{|r|} & \text{if} \quad |r| > \delta \end{cases} \tag{10}$$

3.2 Depth map estimation

First, to say that, LSD SLAM build a semi-dense inverse depth map, only build depth with apparent gradient, and represented as inverse depth. Also there is a assumption that the inverse depth is Gaussian distribution. And there are two kinds of frame, keyframe and reference frame, once a keyframe is determined, the other reference frames is used to refine the depth map of that keyframe, and the new keyframe will be created after a suitable distance.

Here we just discuss the depth map between keyframe and reference frame.

3.2.1 Reference frame selection

Consider a trade-off between precision and accuracy in stereo visual. While a small baseline generate a unique, but imprecise result, a large baseline gives a precise, but with many false result. We use a prob-

abilistic approach of the fact that, small-baseline frames comes out before large-baseline frames.

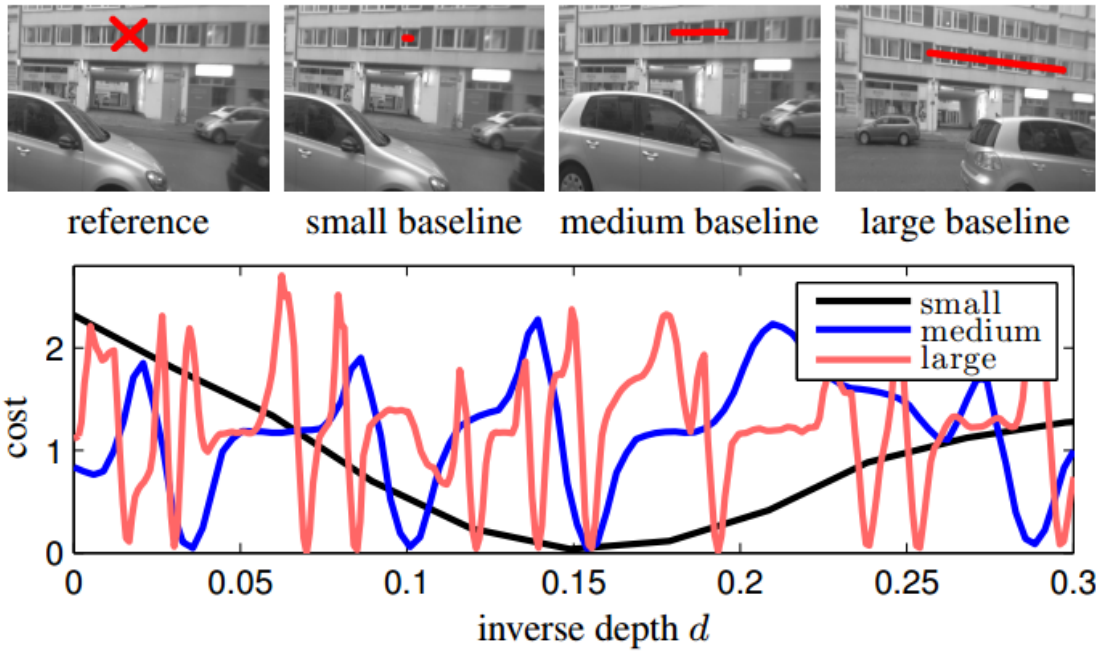

Figure 3.2: variable baseline strero

For every eligible pixel, we select a suitable frame individually, and operate a one-dimensional disparity search. In assumption, the reference frame should be chosen to maximizes the stereo accuracy, while the disparity search range and the observation angle keeps as small as possible.

For the pixel in the new frame(top left)(Figure 2.3), different previous frames are considered, based on how long has the pixel been seen.(older with more yellow). For each pixel we select. We process the following sequence: first to use the pixel in oldest frame to observe, and keeping the disparity search range and observation angle under a certain threshold. If the search is not successful, the age of the pixel increases, so the disparity search in newer frames which the pixel is still visible

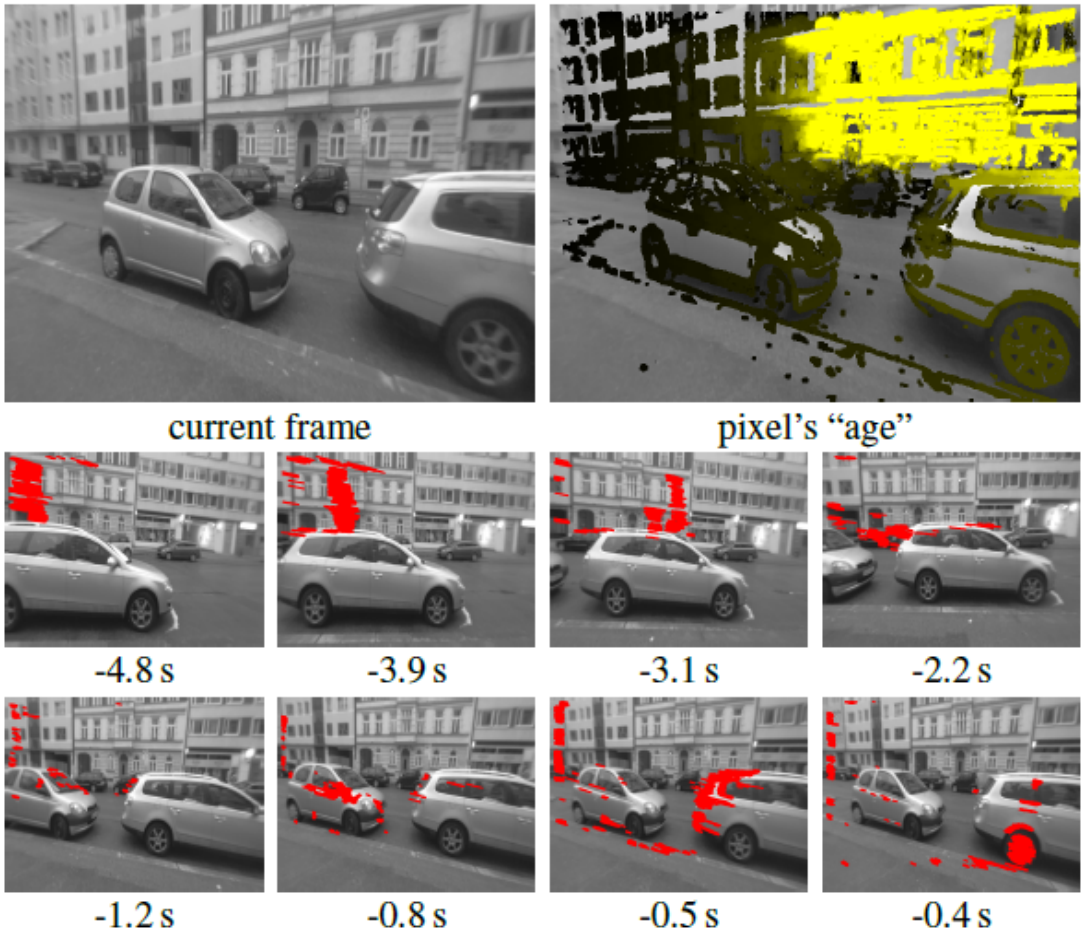

Figure 3.3: variable baseline strero

3.2.2 Matching

We search for the pixel along the epipolar line, then match the pixel to calculate disparity and depth. In this implementation, we use SSD (Sum of Squared Differences) error on five equidistant points on epipolar line. Because this can significantly increase robustness in high-frequent image regions.

$$E_{SSD}(u) = \sum_i [I_1(x_i + u) - I_0(x_i)]^2 \tag{11}$$

3.2.3 Uncertainty estimation

After we get the matching points, in this algorithm, we also get the uncertainty of the points. If we denote the inverse depth map as d^* , we have

$$d^* = d(I_0, I_1, \xi, \pi) \tag{12}$$

Where I_0 and I_1 are the images we consider, and ξ is relative orientation between them, π is the camera calibration model. Then we have the error-variance of d^* is :

$$\sigma_d = J_d \Sigma J_d^T \tag{13}$$

J_d is the Jacobian of d, and the Σ is the covariance of the input-error.

To get a inverse depth, we employ this sequence:

- Computer epipolor line
- Determine the best matching position and the disparity λ^*
- Get the inverse depth d^* from that disparity.

The first two steps contain two kinds of errors:the geometric error, from noise on ξ and π and corrupt first step, and the photometric error from I_0 and I_1 and corrupt second step.

Geometric disparity error

The geometric error is from ξ and π, so we can model and estimate

it from ξ and π. Let assume epipolor line L:

$$L := \left\{ l_0 + \lambda \begin{pmatrix} l_x \\ l_y \end{pmatrix} \mid \lambda \in S \right\} \tag{14}$$

Where λ is disparity in the search interval S,$(l_x, l_y)^T$ is the normalized epipolar line direction and l_0 is a infinite depth point. We consider that only absolute position of this line is under a Gaussian noise ϵ_l. In the program, we keep the searched epipolar line short, so the effect of rotational error is small. A position error ϵ_l on the epipolar line generate a disparity error ϵ_λ. If the epipolar line is parallel to the image gradient, the disparity error will be small , and vice-versa (Figure 2.4).The dash line is the isocurve where the matching point lie.

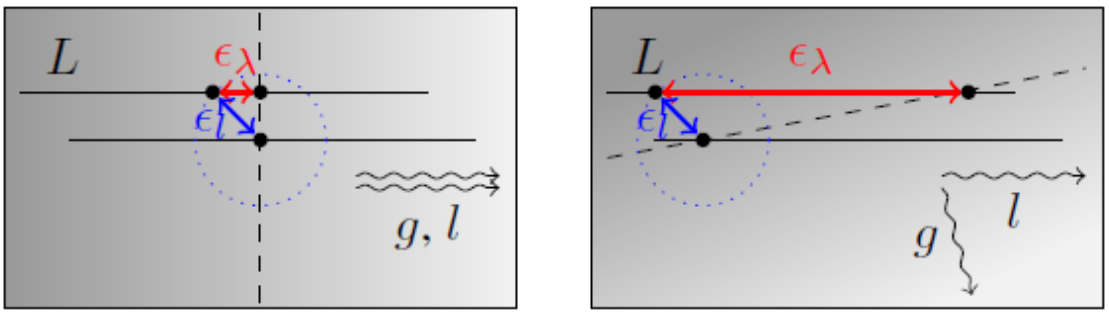

Figure 3.4: geometric error

This problem can be analyzed as : The image restrain the real λ^* on a isocurve, a equal intensity curve. We say this isocurve is locally linear, which means the gradient direction is locally constant, we have

$$l_0 + \lambda \begin{pmatrix} l_x \\ l_y \end{pmatrix} \stackrel{!}{=} g_0 + \gamma \begin{pmatrix} -g_y \\ g_x \end{pmatrix} \qquad \gamma \in \mathbb{R} \tag{15}$$

Where $g := (g_x, g_y)$ is image gradient and g_0 is a point on the

isoline, the effect of noise from image will be analyzed next, now , we assume g and g_0 are noise-free, and solving λ, we have

$$\lambda^*(l_0) = \frac{\langle g, g_0 - l_0 \rangle}{\langle g, l \rangle} \tag{16}$$

Then with equation (13), we can get the variance of the geometric disparity error :

$$\sigma^2_{\lambda(\xi,\pi)} = J_{\lambda^*(l_0)} \begin{pmatrix} \sigma_l^2 & 0 \\ 0 & \sigma_l^2 \end{pmatrix} J^T_{\lambda^*(l_0)} = \frac{\sigma_l^2}{\langle g, l \rangle^2} \tag{17}$$

Photometric disparity error

In a image, if the image gradient is small, then a small image intensity error will have a large effect on the estimated disparity (Figure (2.5) right).In mathematical, the relation can be as following:

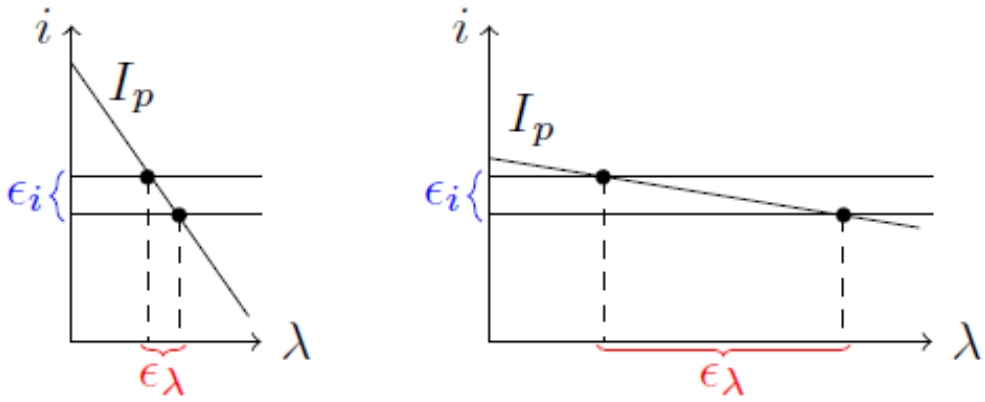

Figure 3.5: photometric error

$$\lambda^* = \min_{\lambda}(i_{ref} - I_p(\lambda))^2 \tag{17}$$

i_{ref} is the reference intensity, and $I_p(\lambda)$ is the image intensity on epipolar line of the disparity λ. We assume after an ideal search on the epipolar line, we can get a good initialization λ_0. The first-order Taylor serious of equation (17) is

$$\lambda^*(I) = \lambda_0 + \Delta\lambda = \lambda_0 + (i_{ref} - I_p(\lambda_0))g_p^{-1} \tag{18}$$

where g_p^{-1} is the gradient of image I_p, so it is one-dimension. Also in this point, only consider the noise in image, and not others. Then we have:

$$\sigma^2_{\lambda(I)} = J_{\lambda^*(I)} \begin{pmatrix} \sigma_i^2 & 0 \\ 0 & \sigma_i^2 \end{pmatrix} J^T_{\lambda^*(I)} = \frac{2\sigma_i^2}{g_p^2} \tag{19}$$

The σ_i^2 is tie variance of the Gaussian noise in the image. This noise is only from the image intensity value, then it is independent from the geometric disparity error.

3.2.4 Pixel to inverse depth conversion

With a small rotation, we can say that the depth d is approximately proportional to the disparity λ, and the variance of the inverse depth $\sigma^2_{d,\text{obs}}$ is

$$\sigma^2_{d,\text{obs}} = \alpha^2 \left(\sigma^2_{\lambda(\xi,\pi)} + \sigma^2_{\lambda(I)}\right) \tag{20}$$

And the coefficient α is different for different pixel, we can assume that the α is

$$\alpha := \frac{\partial d}{\partial \lambda} \tag{21}$$

while ∂d is the length of the inverse depth search interval, and the $\partial \lambda$ is the length of that searched epipolar line segment. Also, it is depends on the pixel's location in the image.

According to that when we use SSD error by more than one points along the epipolar line, there is an upper bound of the matching uncertainty:

$$\sigma_{d,\text{obs}}^2 \leq \alpha^2 \left(\min\{\sigma_{\lambda(\xi,\pi)}^2\} + \min\{\sigma_{\lambda(I)}^2\} \right) \tag{22}$$

3.2.5 Depth observation fusion

When we get depth of a pixel in current image, we need to integrate this depth into the depth map. Our implementation is : If there is no prior hypothesis for this pixel, we build it directly with the observation, if there is prior hypothesis, the new one is incorporated with the prior one, the posterior is calculated as:

$$\mathcal{N}\left(\frac{\sigma_p^2 d_o + \sigma_o^2 d_p}{\sigma_p^2 + \sigma_o^2}, \frac{\sigma_p^2 \sigma_o^2}{\sigma_p^2 + \sigma_o^2} \right) \tag{23}$$

Where $\mathcal{N}(d_p, \sigma_p^2)$ is the prior distribution and $\mathcal{N}(d_o, \sigma_o^2)$ is the noisy observation.

3.2.6 Depth map propagation

We always propagate the inverse depth map from frame to frame,when propagate the inverse depth, we assume the rotation is small, the new

inverse depth d_1 can be approximated as:

$$d_1(d_0) = (d_0^{-1} - t_z)^{-1} \tag{24}$$

while the t_z is the translation along the optical axis, the variance of d_1 is to be :

$$\sigma_{d_1}^2 = J_{d_1}\sigma_{d_0}^2 J_{d_1}^T + \sigma_p^2 = \left(\frac{d_1}{d_0}\right)^4 \sigma_{d_0}^2 + \sigma_p^2 \tag{25}$$

where σ_p^2 is the prediction uncertainty, which corresponds to the prediction step in extended Kalman filter in depth observation fusion step. We get that using a small value of σ_p^2 decrease drift, as it can gradually lock the estimated geometry into place.

In every time, we have only one inverse depth hypothesis for each pixel. If there are two different depth hypothesis try to propagate in the same pixel, we process it as:

- If they are similar, which means lie with 2σ bound, they will both get fusion into new frame according to the equation(23)
- If not, the point further away will be removed

3.2.7 Keyframe selection

If the camera travel a long enough distance or pass a big enought angle, a new keyframe will be created . We set a threshold to determine when to create a keyframe:

$$\text{dist}(\xi_{\mathbf{ji}}) := {\xi_{\mathbf{ji}}}^T \mathbf{W} \xi_{\mathbf{ji}} \tag{26}$$

Here, $\mathbf{W}$ is a matrix represent the weights. We create each keyframe with its mean inverse depth is one.

3.3 Map optimization

Finally, we have a map based on every keyframe, and we optimize the global map using this equation by the g2o package:

$$E(\xi_{W_1}...\xi_{W_n}) := \sum_{(\xi_{ji},\sum_{ji})\in\varepsilon} (\xi_{ji} \circ \xi_{W_i}^{-1} \circ \xi_{W_j})^T \Sigma_{ji}^{-1} (\xi_{ji} \circ \xi_{W_i}^{-1} \circ \xi_{W_j}) \quad (3.2)$$

Chapter 4

CNN SLAM

As we know in last chapter, to build a depth map, some pixels are assumed without any prior information. Here, as the deep learning technical develops so quickly in now days, it is possible for us to provide a better prior estimation to the algorithm then randomly generate. Depth estimation has a long history in computer vision, as introduce in previous chapter, here I show a method generate estimation from only a signal image.

4.1 Introduction to CNN

Neural Networks (NN), or more precisely *Artificial Neural Networks* (ANN), is a kind of Machine Learning algorithms which are an information processing inspired by the approach of processing of human brain.

A standard *Neural Network* (NN) consists of many connected processors which are called *neurons*, each of them producing a sequence of real-valued operations. Input neurons get activated through sensors based on the environment, other neurons get activated through

weighted connections from previous active neurons. Some neurons may react the environment by triggering actions. Depending on what problem is dealing with and how the neurons are connected, such process may require long causal chains of computation, where each stage transforms (often in a non-linear way) the aggregate activation of the network.

Neural networks is a generic name for a large class of machine learning algorithms, including but not limited to: perceptrons, fully connected neural networks, convolutional neural networks, recurrent neural networks, long short term memory neural networks and many more. Most of them are trained by an algorithm called *back-propagation*.

CNNs are hierarchical neural networks which, in another words, has one or more layers of convolution units, consists of simple and complex cells in the primary visual cortex.

CNN utilize layers with convolutional filters that are processed on local features. Originally invented for computer vision. In the previous studies, CNN has been shown as a good pattern recognition ability and robust classifier, with the ability to generalize in making decisions based even on imprecise input image. CNN models have subsequently been shown to be effective for NLP and have got excellent results in semantic parsing , search query retrieval, sentence modeling , and other traditional NLP tasks.

CNNs changes in how convolutional and subsampling layers are created and how the network are trained.

They mainly have three important architectural features:

- **local Connectivit**: Neurons in one layer are only connected to neurons in the next layer that are spatially close to them.

- **shared weights**: This is the concept that makes CNNs "convo-

lutional".

- **pooling and ReLU**: CNNs have two non-linearities: pooling layers and ReLU functions.

4.2 Basic concept of CNN

As shown in figure 4.1, the left matrix is a image, the number represents gray scale or color value, if we use a matrix in middle (filter) to take the element wise product on left matrix, for example, if the middle matrix is attached on the top left of the left matrix, we will get

$$10*1+10*1+10*1+10*0+10*0+10*0-10*1-10*1-10*1=0 \quad (4.1)$$

As on the top left element of the right matrix, then we do this operation from left to right, top to bottom, we will get the right matrix. As we can see in right matrix, there is a obvious edge in the middle.

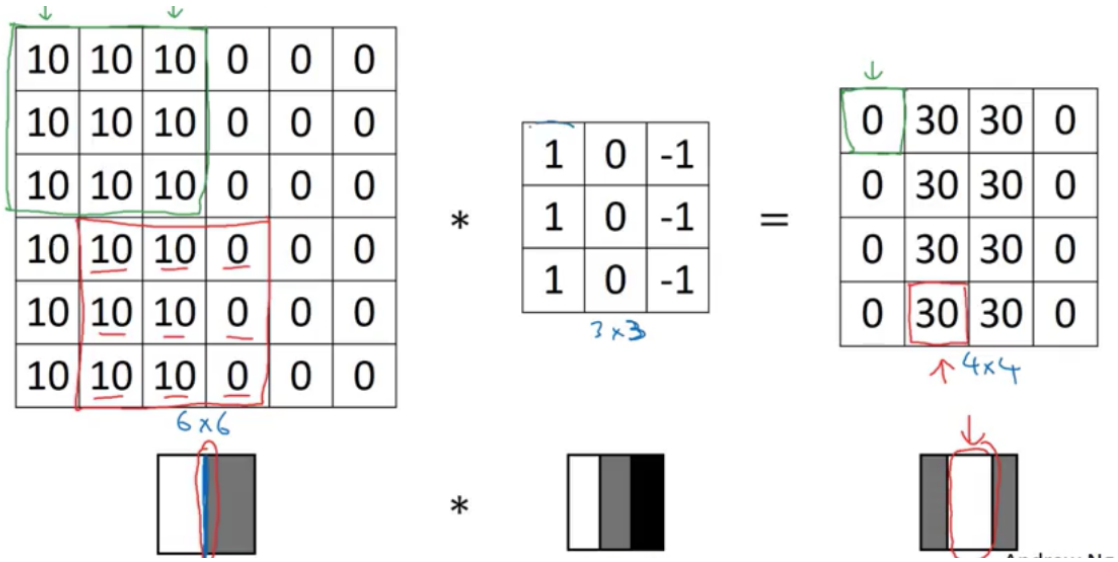

Figure 4.1: convolution operation

If we have a common RGB image, then we can make the filter be a $3 * 3 * 3$ matrix. Then we can use this filter to process a 3 channels RGB image. as shown in figure 4.2

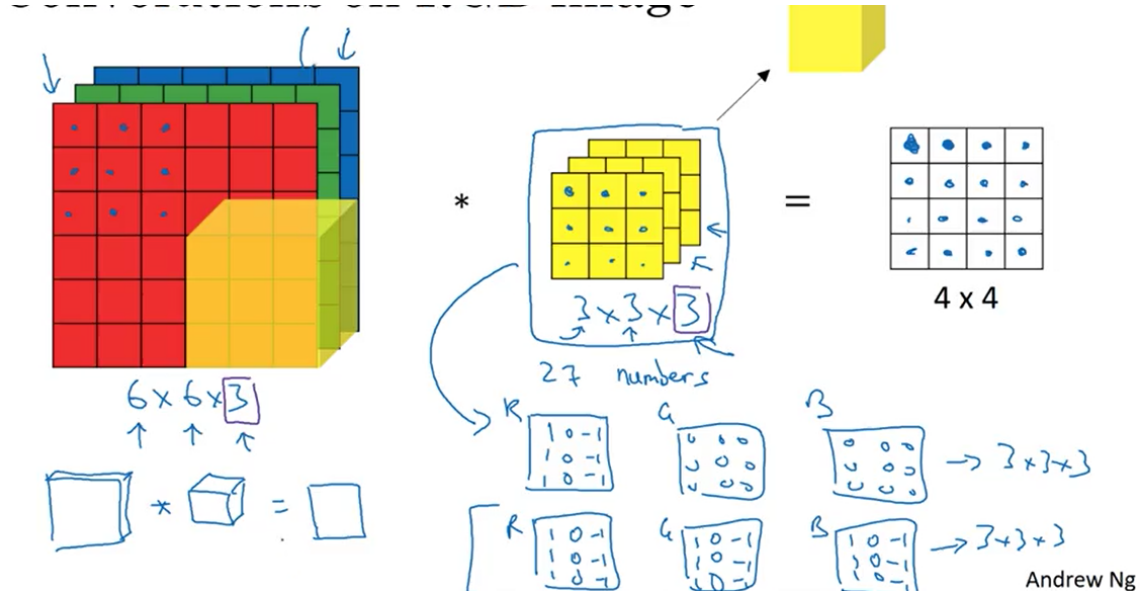

Figure 4.2: 3 channels convolution

After that, we will have pooling layer to process the output matrix

in last matrix, such as in figure 4.3. This is a Max pooling.

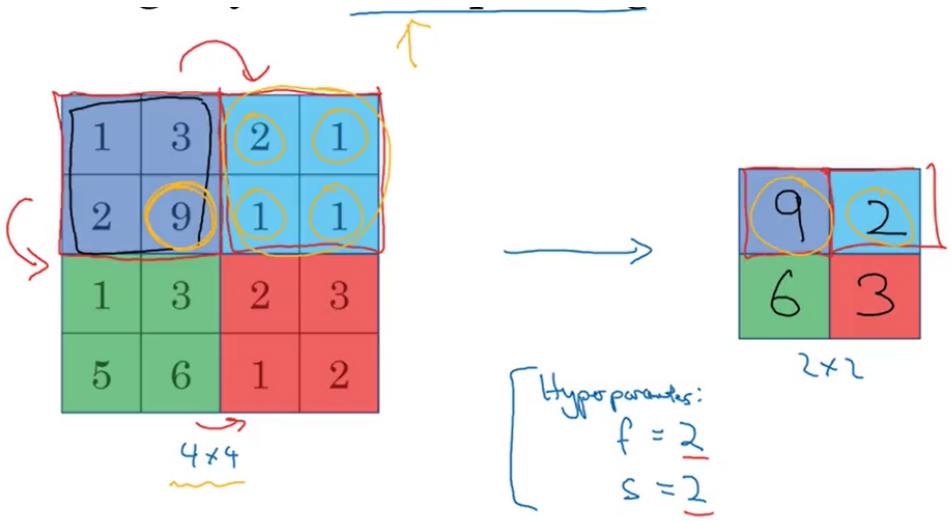

Figure 4.3: pooling

So in a convolutional network, we will have three types of layer

- Convolution (Conv)
- Pooling (Pool)
- Fully connected (FC)

Here we show a example of CNN based on LeNet-5 figure 4.4, LeNet-5 4.5 refdeveloped by Yann Lecun who is a pioneer in the machine learning field gets a very good result in character recognition, also know as Optical Character Recognition (OCR), is a very important and basic field in CNN technical.

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208
POOL1	(14,14,8)	1,568	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

Figure 4.4: CNN example

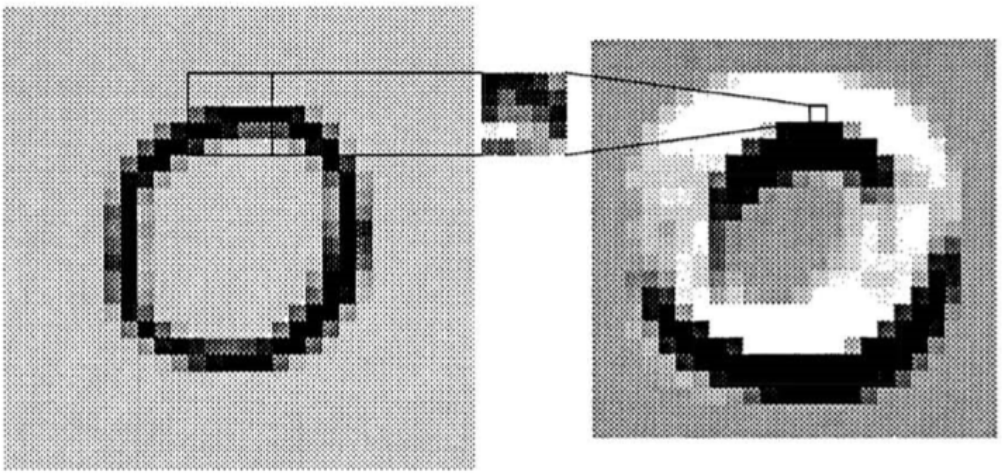

Figure 4.5: OCR example

Input image (left), and resulting feature map (right), the feature map is created by scanning a input image with a neuron which has a local receptive field. As shown, white indicates -1, black indicates +1.

4.3 Unsupervised Monocular Depth Estimation with Left-Right Consistency

A convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks,it shows a great performance in many fields of computer vision.

In traditional, some method are implemented to get a depth estimation,but they need a patch-based model or a hand crafted features to begin. Also these approached rely on a high quality ground truth depth at training time. Here, I introduce a method to train data without ground truth, because sometimes, the ground truth is not easy to achieve due to some limitation. For example, the environment is too complex, also the cost of a depth sensor is much more than the optical camera.

With a monocular image I, our goal is building a function f that predict a depth image for each pixel $\hat{d} = f(I)$. Most of existing approaches process this problem as a supervised one, they use the each pixel color and corresponding depth values as input when train. Obviously, it is not economic or practical to get a large ground depth data. Here, we treat this problem as an image reconstruction problem. For example, with a stereo camera, if we can know how to reconstruct a image in left camera only from the image in right camera, then we can get a disparity and to build a depth. While we training, we process two images I^l, I^r from both sides of camera at the same moment, then to find the dense correspondence field d^r, that could help us to reconstruct right image from left image. We denote this reconstructed image as $\tilde{I}^r$. At the same time, we also reconstruct left image from right image, and denoted as $\tilde{I}^r$. So we have $\tilde{I}^r = I^l(d^r)$ and $\tilde{I}^l = I^r(d^l)$.

And d correspond to the disparity, a value that the model should learn to predict. With the known parameter of the stereo camera, such as baseline b and the camera focal length f, then we should be able to recover the depth $\hat{d} = bf/d$.

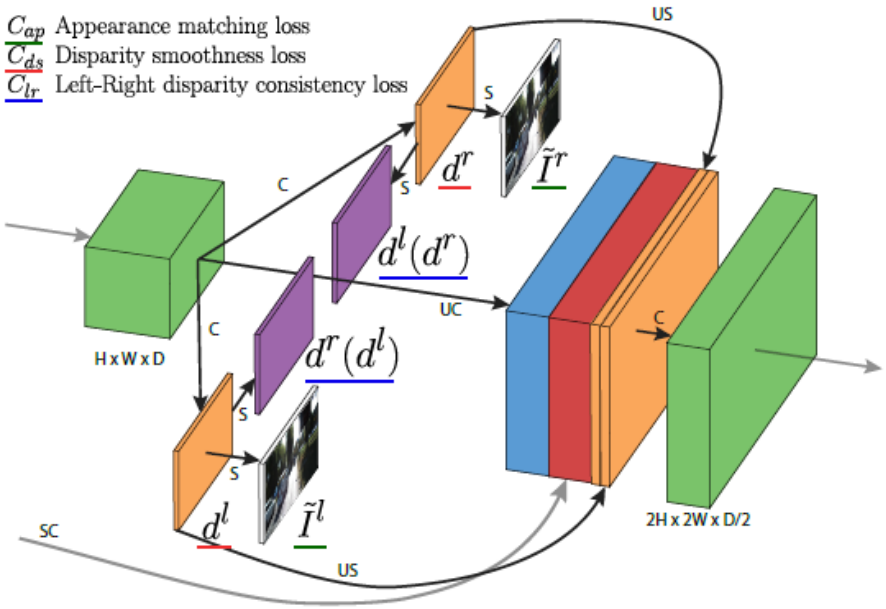

Figure 4.6: monodepth overview

The key of this method is that get both left-to-right and right-to-left disparities, only based on the left input image, and get a good depth estimation by enforcing them to be consistent.

As shown in the Figure 4.7 the naive network try to only generate disparities aligned on the right image. But, we want the disparity aligned on the input, the left image. We solve this by training the network to generate the predicted disparity map from both image by using the opposite image as input. So the right image as input is only used while training. Enforce these two disparities to be consistency can leads to a more accurate result. This convolutional architecture is inspired by DispNet, but some feature is modified that allow us to train

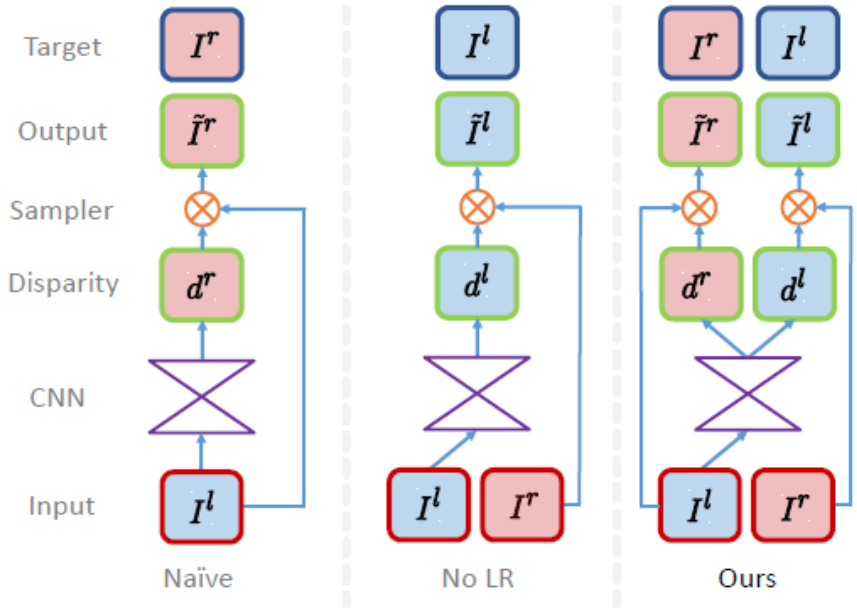

Figure 4.7: monodepth network

without the ground truth depth. This network contains two parts,an encoder (from cnv1 to cnv7b) and decoder (from upcnv7), shown as Figure 4.9.

4.3.1 Training loss

We define a C_s as a loss at each output s, and set the total loss as $C = \sum_{s=1}^{4} C_s$, the loss contain three main terms,

$$C_s = \alpha_{ap}(C_{ap}^l + C_{ap}^r) + \alpha_{ds}(C_{ds}^l + C_{ds}^r) + \alpha_{lr}(C_{lr}^l + C_{lr}^r), \tag{4.2}$$

where C_{ap} is the reconstructed image similarity to the corresponding input, C_{ds} is smooth disparities, and the C_{lr} is the left and right disparities to be consistent.

4.3.2 Appearance Matching Loss

First, we use a image sampler as a image formation from the STN(spatial transformer network). Then we united an L_1 and single scale SSIM (structural similarity index) as cost C_{ap}.

$$C_{ap}^{l} = \frac{1}{N}\sum_{i,j} \alpha \frac{1 - \mathrm{SSIM}(I_{ij}^{l}, \tilde{I}_{ij}^{l})}{2} + (1-\alpha)\left\| I_{ij}^{l} - \tilde{I}_{ij}^{l} \right\|. \tag{4.3}$$

where I_{ij}^{l} is the input image, and $\tilde{I}_{ij}^{l}$ is its reconstruction, N is the number of pixels, here we set α=0.85.

4.3.3 Disparity Smoothness Loss

We set a goal that the disparities should be locally smooth. As there often appear depth discontinuities at image gradients, we give this cost with an edge-aware using the image gradients ∂I,

$$C_{ds}^{l} = \frac{1}{N}\sum_{i,j} \left|\partial_x d_{ij}^{l}\right| e^{-\left\|\partial_x I_{ij}^{l}\right\|} + \left|\partial_y d_{ij}^{l}\right| e^{-\left\|\partial_y I_{ij}^{l}\right\|}. \tag{4.4}$$

4.3.4 Left-Right Disparity Consistency Loss

Here is the most important, the disparity consistency between left-right and right-left.

$$C_{lr}^{l} = \frac{1}{N}\sum_{i,j} \left| d_{ij}^{l} - d_{ij+d_{ij}^{l}}^{r} \right|. \tag{4.5}$$

And here is the result:

Figure 4.8: monodepth result

"Encoder"						
layer	**k**	**s**	**chns**	**in**	**out**	**input**
conv1	7	2	3/32	1	2	left
conv1b	7	1	32/32	2	2	conv1
conv2	5	2	32/64	2	4	conv1b
conv2b	5	1	64/64	4	4	conv2
conv3	3	2	64/128	4	8	conv2b
conv3b	3	1	128/128	8	8	conv3
conv4	3	2	128/256	8	16	conv3b
conv4b	3	1	256/256	16	16	conv4
conv5	3	2	256/512	16	32	conv4b
conv5b	3	1	512/512	32	32	conv5
conv6	3	2	512/512	32	64	conv5b
conv6b	3	1	512/512	64	64	conv6
conv7	3	2	512/512	64	128	conv6b
conv7b	3	1	512/512	128	128	conv7

"Decoder"						
layer	**k**	**s**	**chns**	**in**	**out**	**input**
upconv7	3	2	512/512	128	64	conv7b
iconv7	3	1	1024/512	64	64	upconv7+conv6b
upconv6	3	2	512/512	64	32	iconv7
iconv6	3	1	1024/512	32	32	upconv6+conv5b
upconv5	3	2	512/256	32	16	iconv6
iconv5	3	1	512/256	16	16	upconv5+conv4b
upconv4	3	2	256/128	16	8	iconv5
iconv4	3	1	128/128	8	8	upconv4+conv3b
disp4	3	1	128/2	8	8	iconv4
upconv3	3	2	128/64	8	4	iconv4
iconv3	3	1	130/64	4	4	upconv3+conv2b+disp4*
disp3	3	1	64/2	4	4	iconv3
upconv2	3	2	64/32	4	2	iconv3
iconv2	3	1	66/32	2	2	upconv2+conv1b+disp3*
disp2	3	1	32/2	2	2	iconv2
upconv1	3	2	32/16	2	1	iconv2
iconv1	3	1	18/16	1	1	upconv1+disp2*
disp1	3	1	16/2	1	1	iconv1

Table 1: Our network architecture, where **k** is the kernel size, **s** the stride, **chns** the number of input and output channels for each layer, **in**put and **out**put is the downscaling factor for each layer relative to the input image, and **input** corresponds to the input of each layer where $+$ is a concatenation and $*$ is a $2\times$ upsampling of the layer.

Figure 4.9: monodepth model

Chapter 5

Result

Due to the official LSD-SLAM is only generate gray scale point, so first step, we need to give every point RGB information, in the program, there is a class called "Frame", it is the most important object in this program, after read the code, we can know that at the every beginning, the program read the image from camera by Opencv, this means, we can add the a data type with color by adding some syntax beside where it read image gray scale, according the Opencv documentation, it should be like as:

```
1 cv::Mat imageDistRGB = cv::imread(files[i], CV_LOAD_IMAGE_COLOR);
```

Also, the data type is CV_8UC3 beside the official CV_8U type, where C3 means 3 channels, which is RGB (or actually BGR) data here. And than, at every places where define the gray scale, we add this variable. Finally, we can get a colored point cloud.

To get a reconstruction result, there will be a denoise step should be operated. I choose a software "CloudCompare" for convenient, there has been many method to process the point cloud integrated in this software. We use Noise filter and SOR (Statistical Outlier Removal)

filter iteratively to cancel different kinds of noise.

After denoise, a comparison between our results and ground truth should be taken to verify the performance of the SLAM approach. Due to scale problem, it is very hard to align our result to ground truth automatically. So first, I build a mesh based on the point cloud result, then choose three point with most confidence to match to the corresponding points in the ground truth, also in this step, scale adjustment is allowed. Then the mesh based on our result will be approximate same scale to the ground truth. Then a feature in CloudCompare called "Fine registration with ICP(Iterative closest point)" can be used. After this operation, our result will be align on the ground truth as close as possible. Then we can calculate distance from our point cloud to ground truth mesh to verify if the result is acceptable. If yes, the mesh based on our result should be acceptable. The ground truth will be download from Google Earth and import in CloudCompare as geographic coordinate, so in most case, the altitude of the ground truth is about 1000 times more than normal, so this data will be adjusted.

5.1 Result of original LSD-SLAM

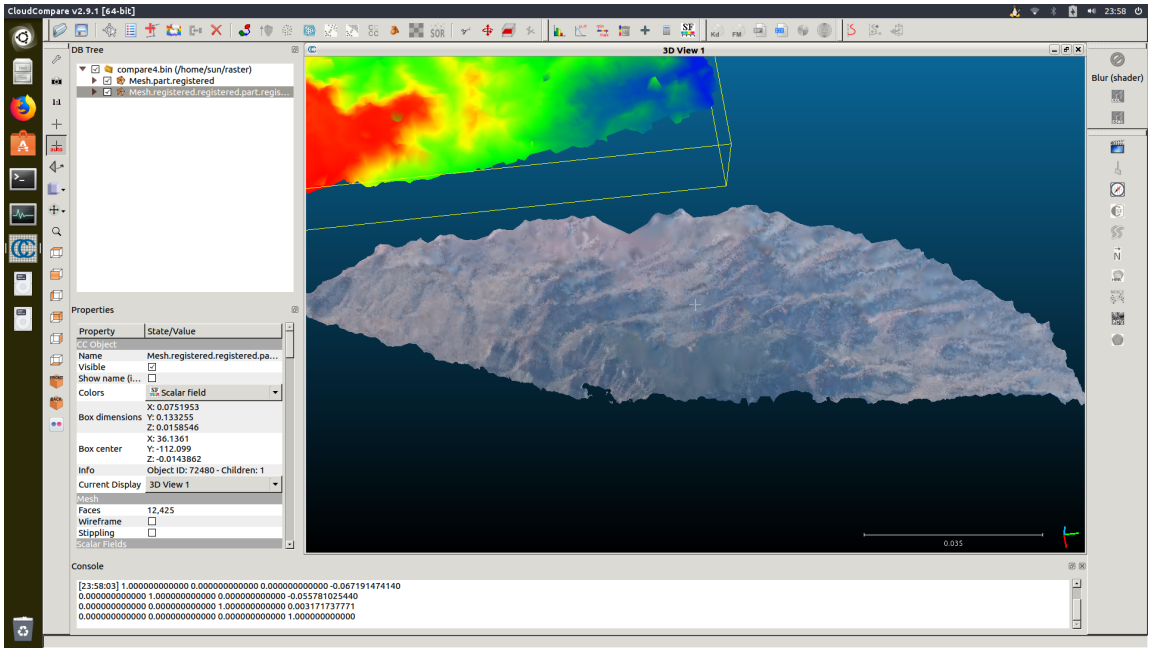

Figure 5.1: grand canyon

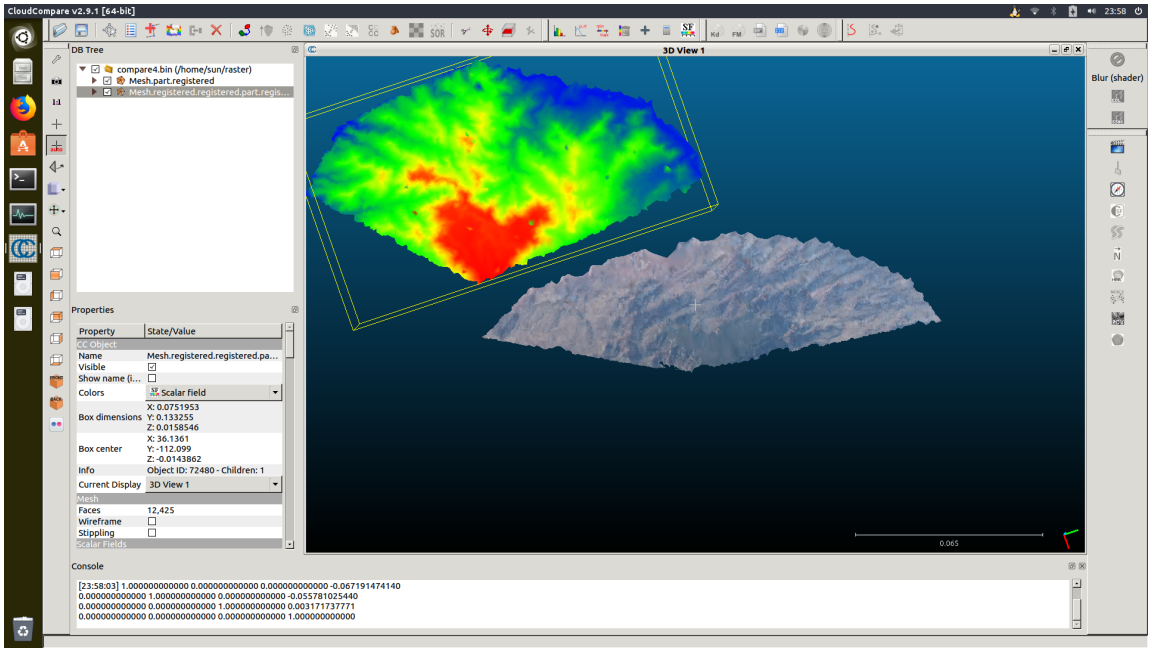

Figure 5.2: grand canyon compare with ground truth

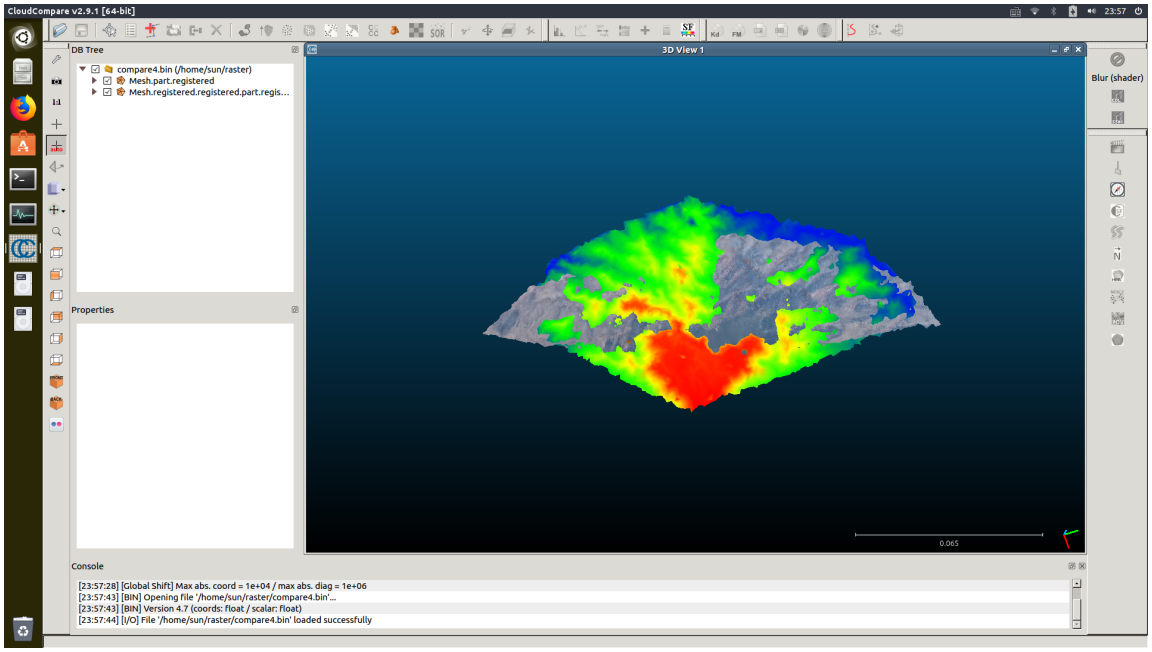

Figure 5.3: grand canyon reconstruct error between ground truth

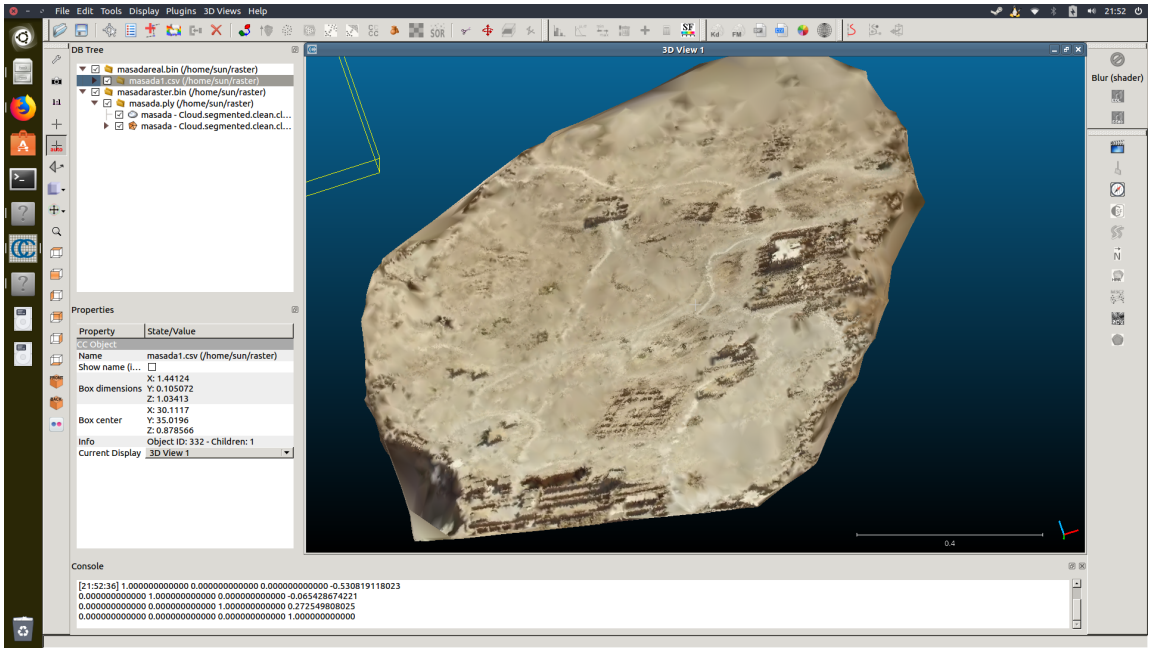

Figure 5.4: masada1

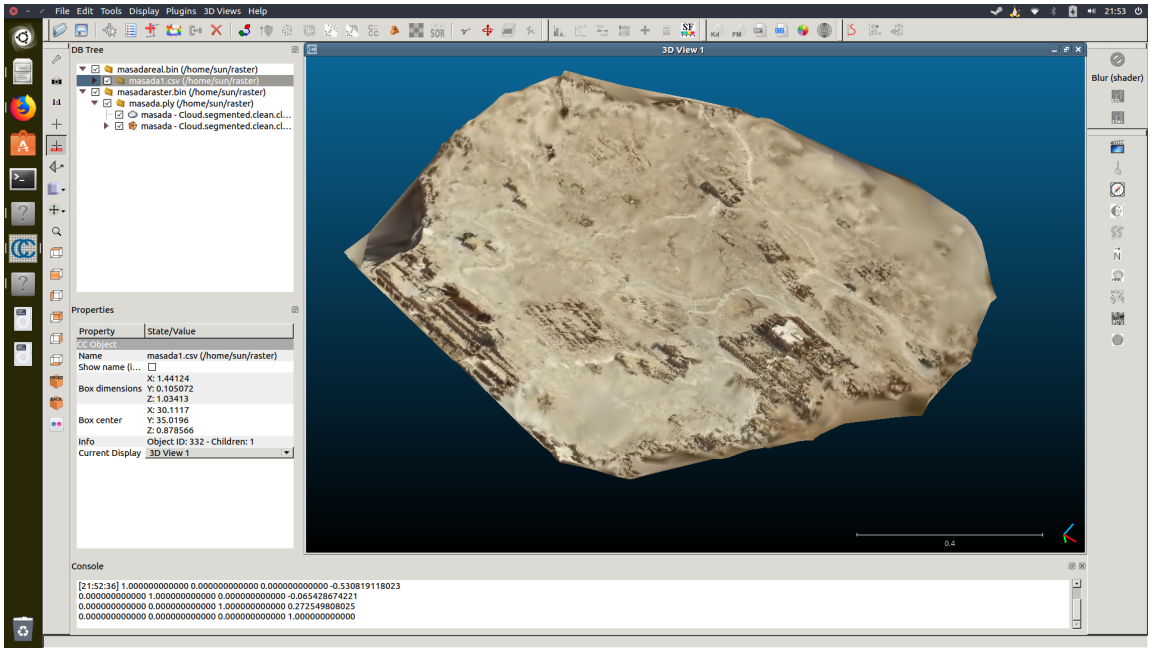

Figure 5.5: masada2

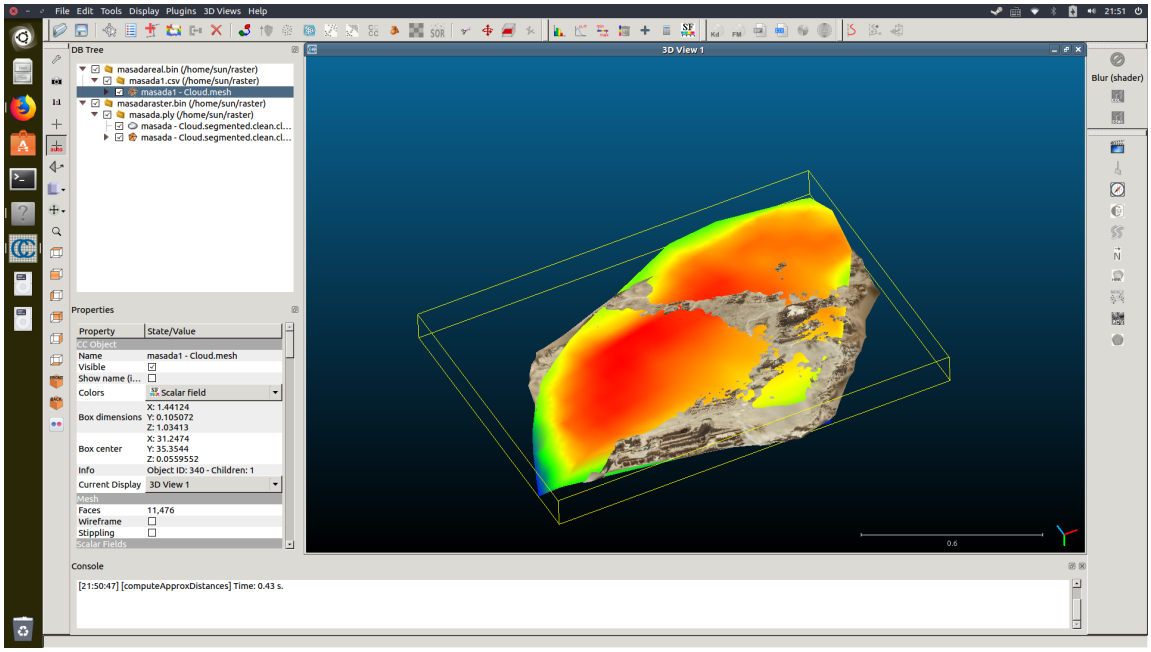

Figure 5.6: masada reconstruct error between ground truth

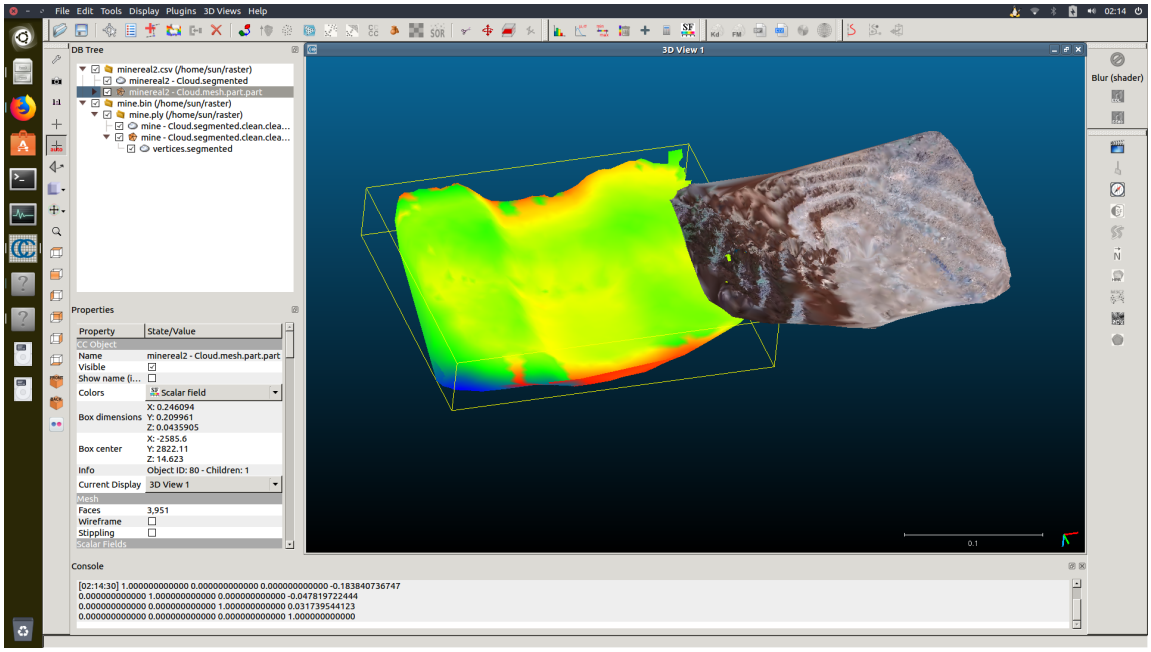

Figure 5.7: mine compare with ground truth

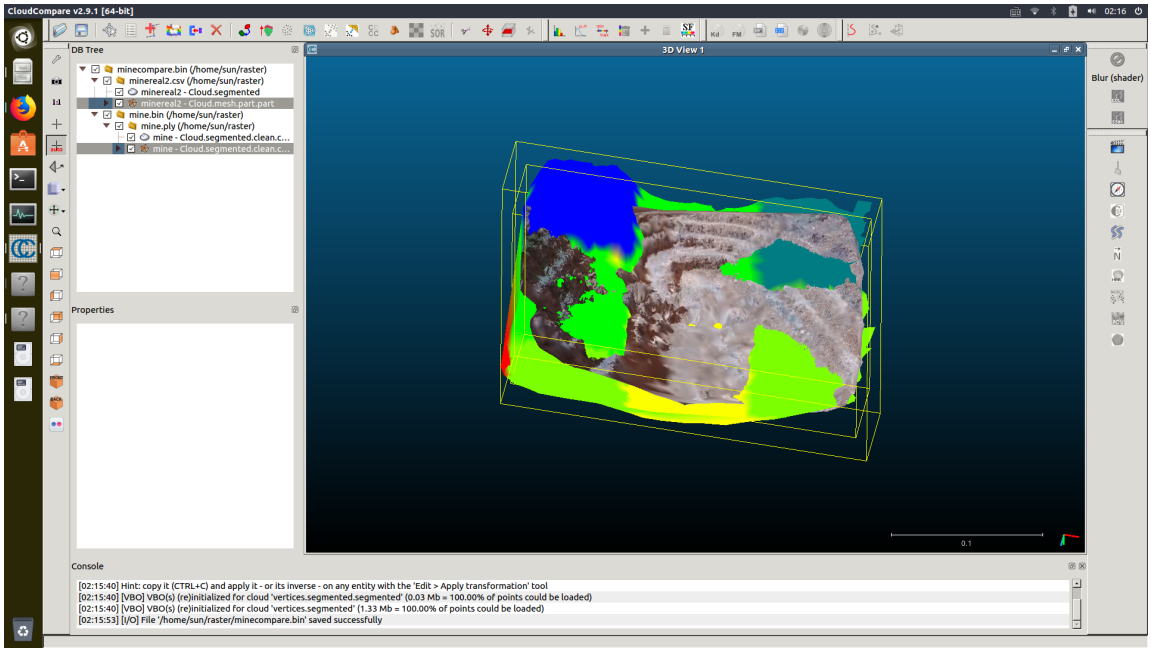

Figure 5.8: mine reconstruct error between ground truth

Due to the monocular SLAM is lack of the real scale, so here only present the relative error respect to the proportional ground truth.

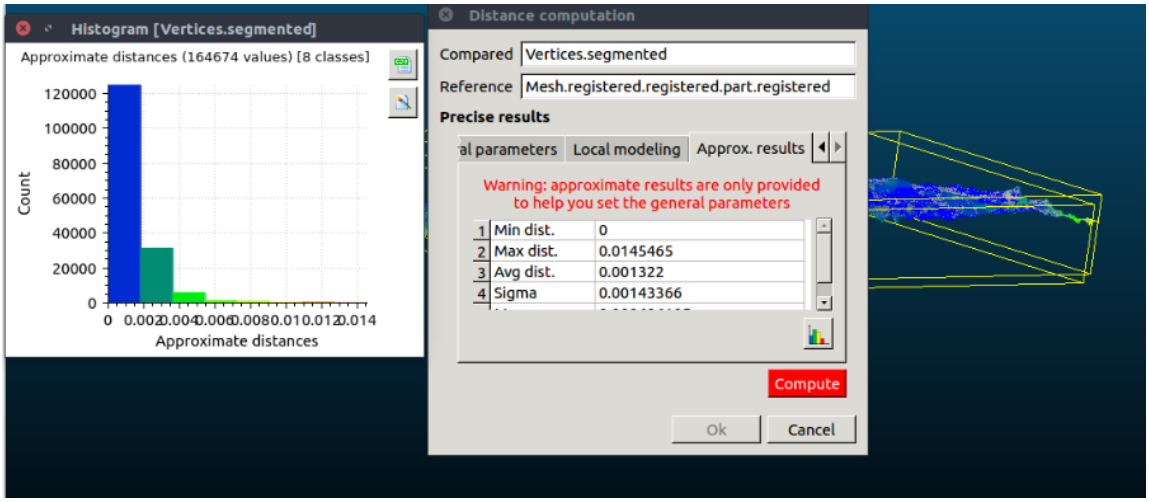

Figure 5.9: Grand Canyon error distribution

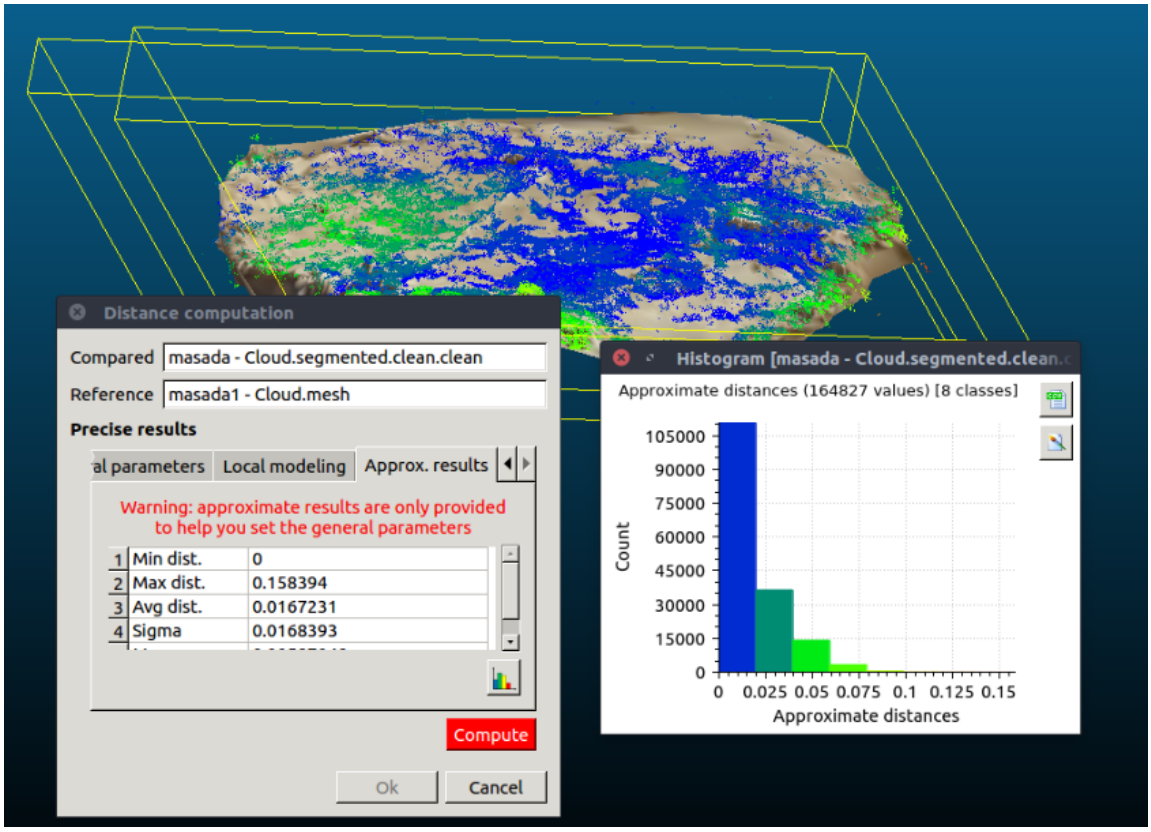

Figure 5.10: Masada error distribution

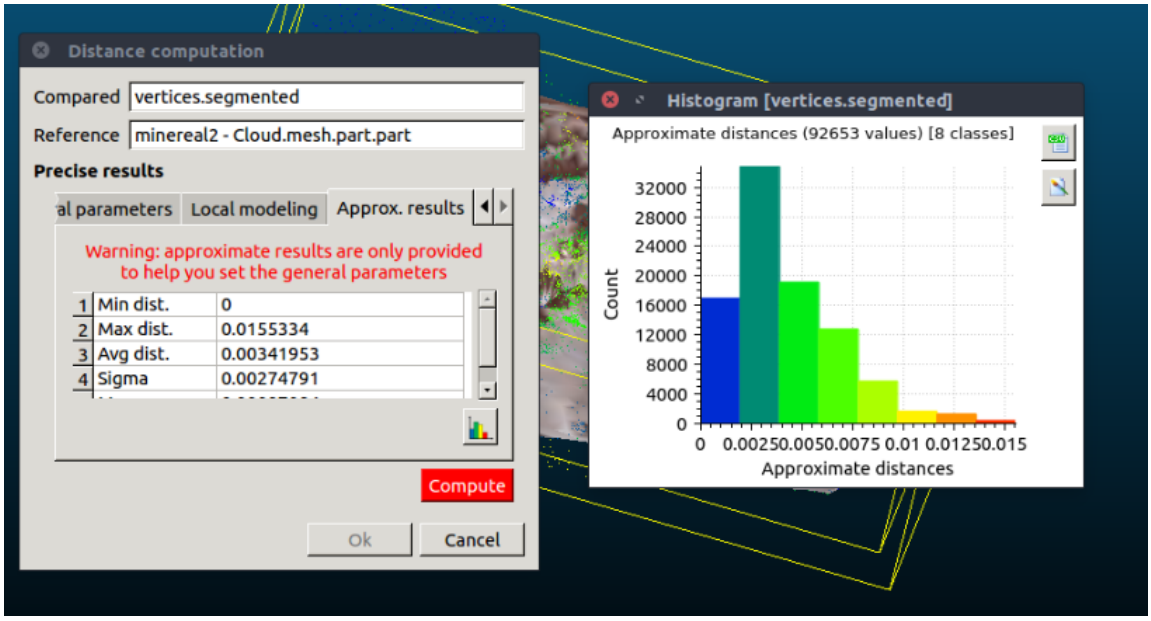

Figure 5.11: mine error distribution

	ground truth average hight	average error	relative error
grand canyon	0.0155	0.001322	8.5%
mine	0.048	0.003	6.2%
masada	0.8826	0.017	1.9%

Table 5.1: good result of LSD-SLAM

5.2 Result of LSD-SLAM with CNN depth estimation

As we can see, we have an network that can provide a acceptable depth estimation, this can be used in the LSD-SLAM algorithm. As talked in the chapter 3, there is a step, that we decide if a depth information of a pixel exist or not. Then here, we have a depth map for every pixel of the image, then we can provide this depth to the algorithm, then this depth information could be fusion with the observation depth information, this step will reduce the noise from the original algorithm. Then we can see the result. The official Monocular depth estimation project is called "Monodepth", and as most of deep learning project, this is written in Python, and LSD-SLAM is written in C++. The problem is that how to integrate them together. According to Python C++ API, it shows that Python C++ interpreter could only be called once in a program, if it is called more than once, there will be a segmentation fault, and this is a official bug that may not be solved in my level, so, to verify the ideal of this thesis, I just generate the depth map frame by frame in a video, and store every depth map, then when I run the LSD-SLAM, I read these depth map depends on the frame ID, because that if we set rate of LSD-SLAM equal to 0, this means that it will process every frame, so the frame ID in LSD-SLAM will mate the real frame ID in video, this means the ID of depth map will correspond to the ID of original image read by the LSD-SLAM, in this way, we can provide the depth map to the SLAM, but this is a prototype to verify the performance of this ideal, there should be a way to use C++ API of the tensorflow to ignore the Python API bug.

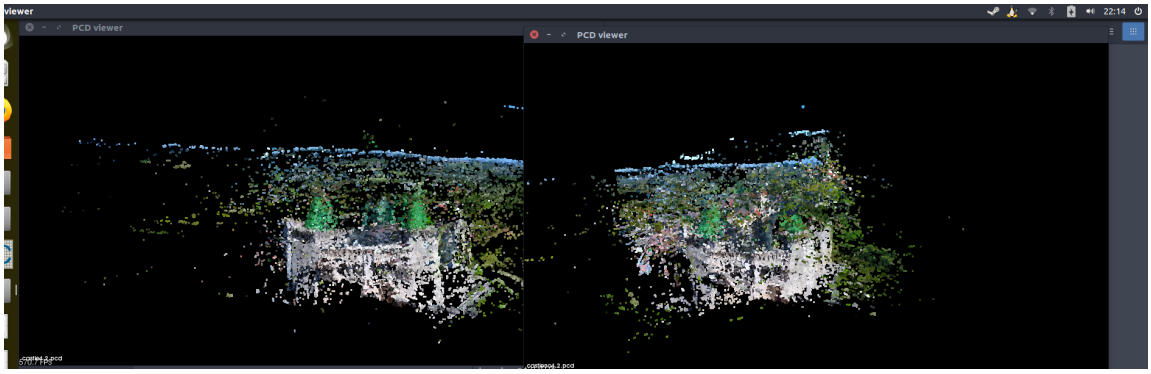

Figure 5.12: castle case:compare between original SLAM(right) and CNN depth estimation SLAM(left) 1

Figure 5.13: castle case:compare between original SLAM(right) and CNN depth estimation SLAM(left) 2

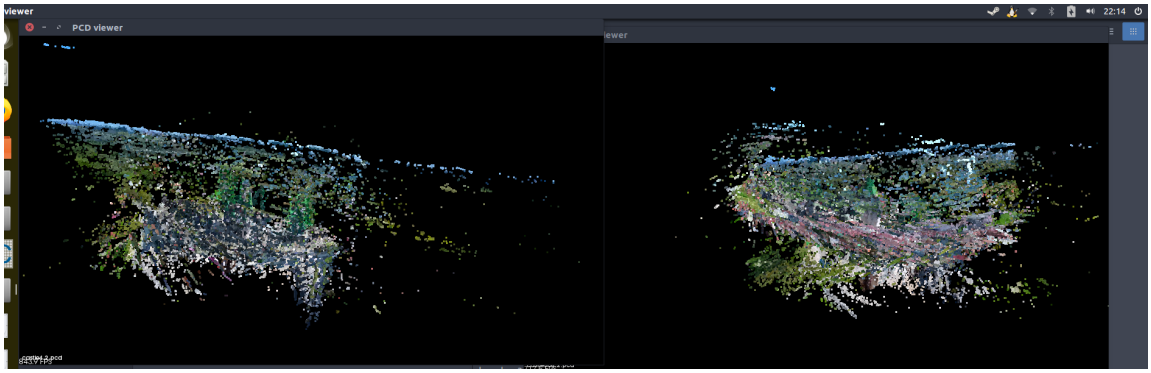

Figure 5.14: castle case:compare between original SLAM(right) and CNN depth estimation SLAM(left) 3

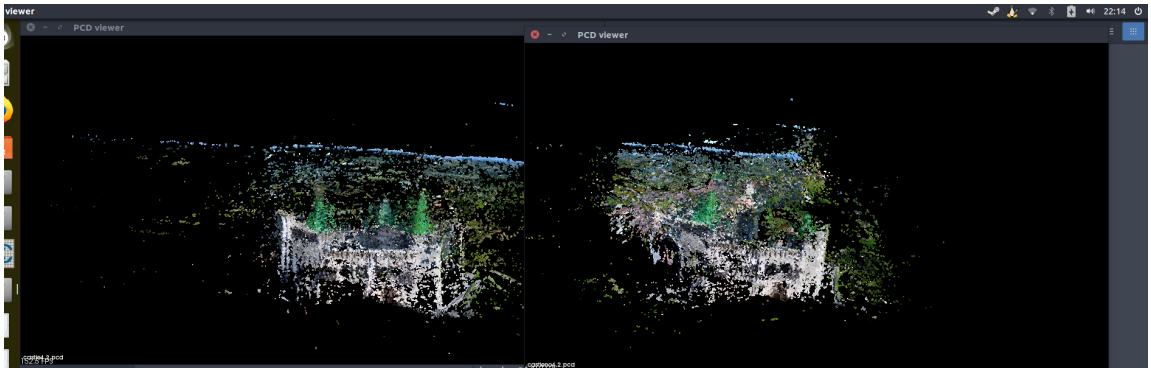

Figure 5.15: castle case:compare between original SLAM(right) and CNN depth estimation SLAM(left) 4

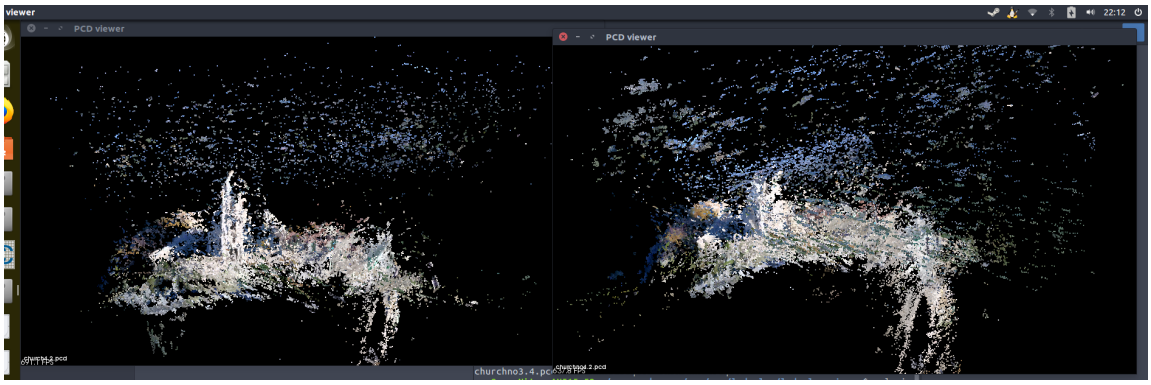

Figure 5.16: church case:compare between original SLAM(right) and CNN depth estimation SLAM(left) 1

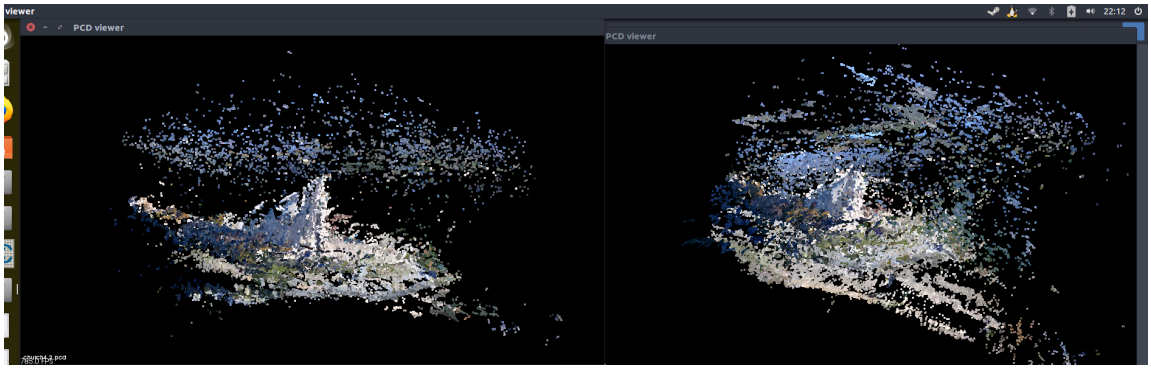

Figure 5.17: church case:compare between original SLAM(right) and CNN depth estimation SLAM(left) 2

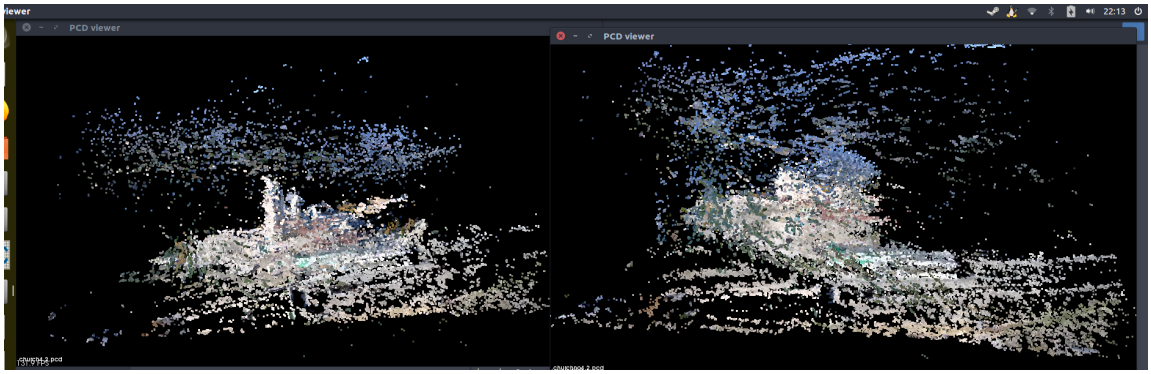

Figure 5.18: church case:compare between original SLAM(right) and CNN depth estimation SLAM(left) 3

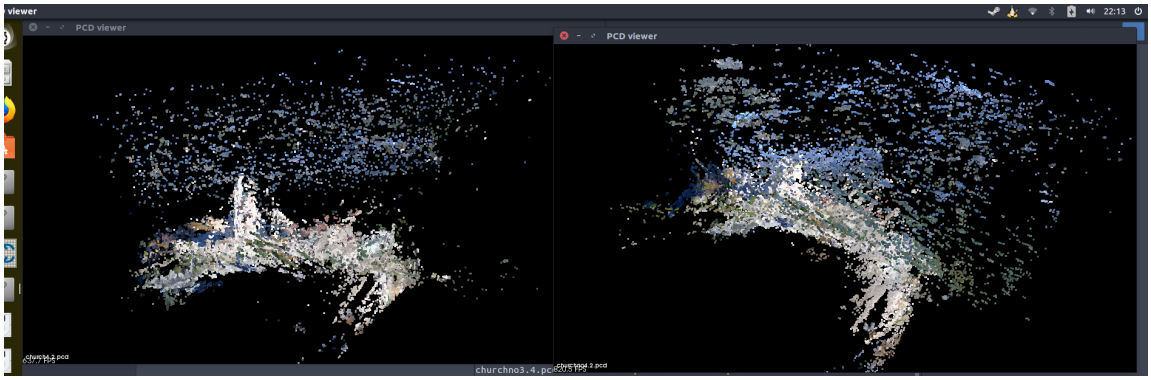

Figure 5.19: church case:compare between original SLAM(right) and CNN depth estimation SLAM(left) 4

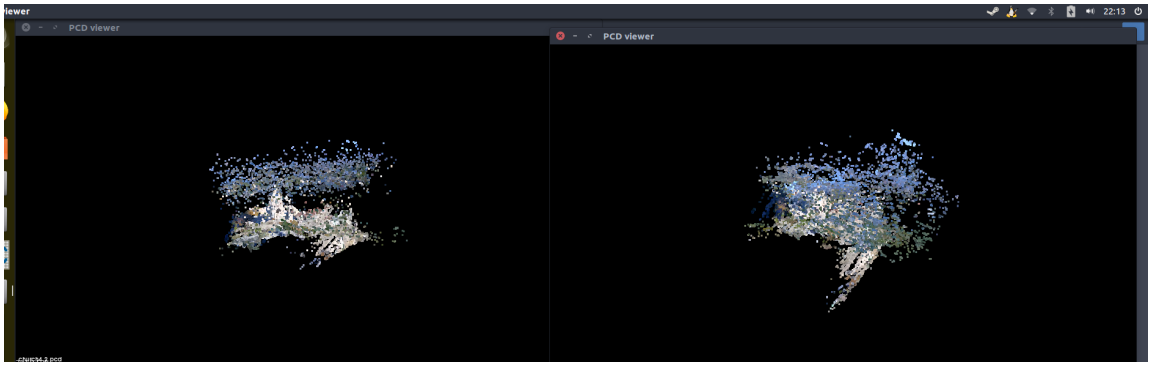

Figure 5.20: church case:compare between original SLAM(right) and CNN depth estimation SLAM(left) 5

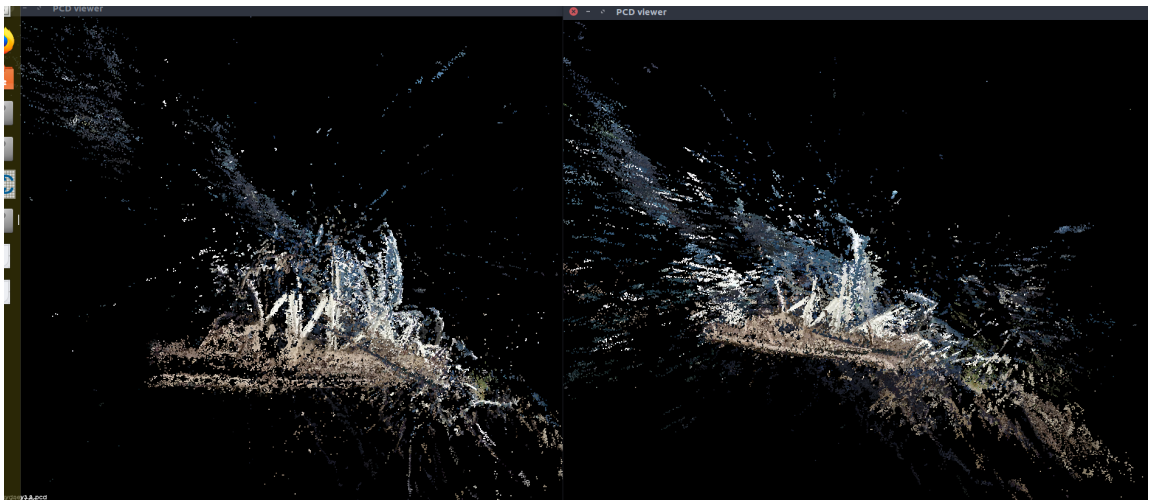

Figure 5.21: Sydney Opera House case:compare between original SLAM(right) and CNN depth estimation SLAM(left) 1

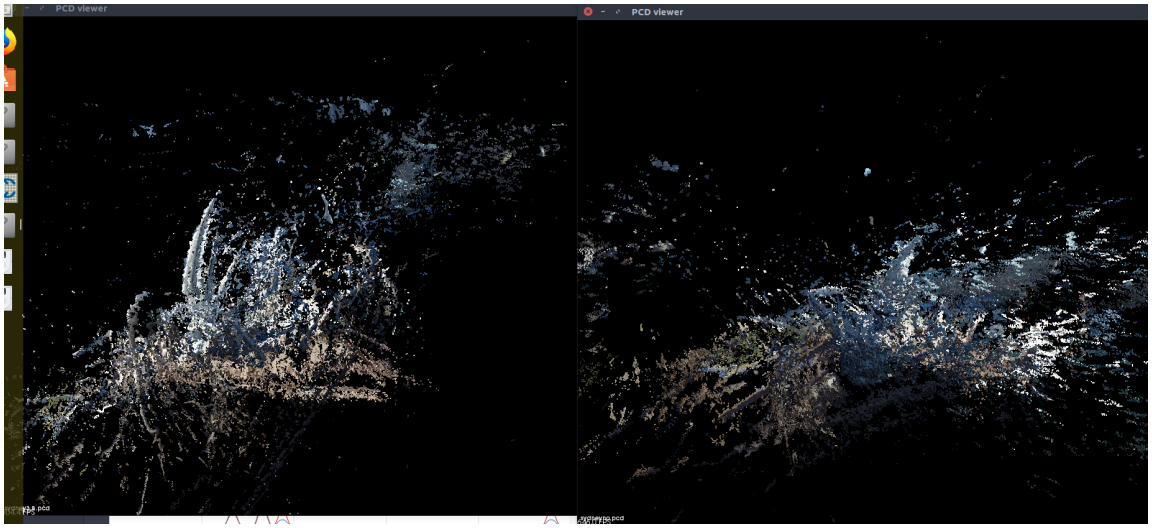

Figure 5.22: Sydney Opera House case:compare between original SLAM(right) and CNN depth estimation SLAM(left) 2

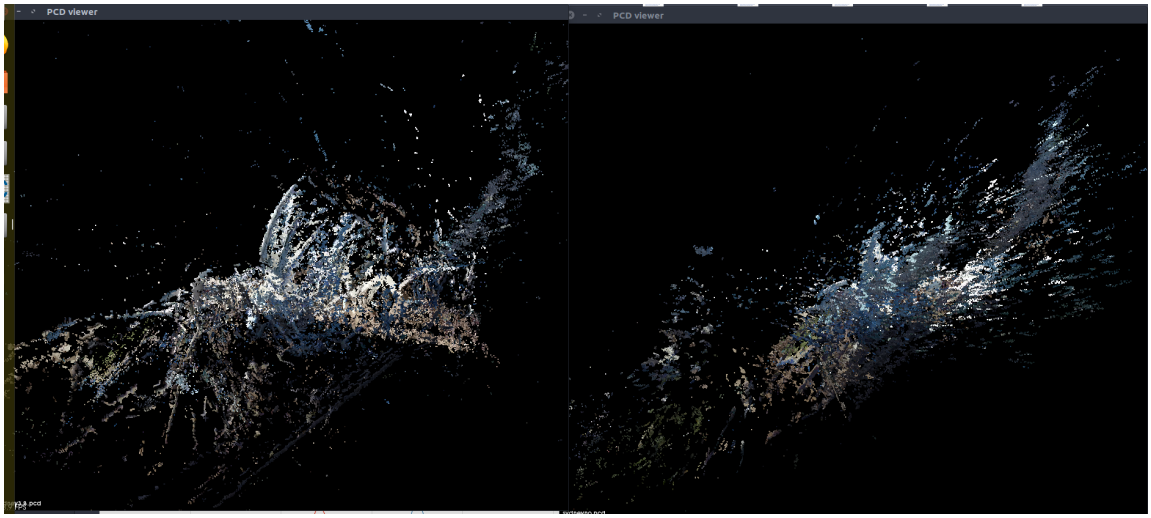

Figure 5.23: Sydney Opera House case:compare between original SLAM(right) and CNN depth estimation SLAM(left) 3

Also due to lack of real scale, we scale the point cloud to the approximate real scale and compare to the ground truth, then get the following result

	original slam average error	cnn slam average error
Memphis pyramid	55m	22m
Sydney Opera House	33.32m	22.7m

Table 5.2: compare between original SLAM and CNN depth estimation SLAM

Chapter 6

Conclusion and future work

As we can see, to reconstruct by SLAM approach, the direct method is better than feature-based method, because the direct method can provide more detail of the structure, and after test several cases, we can say that the direct SLAM can produce a acceptable result if we can control the UAV trajectory strictly. Once there is some rotation, the point cloud will be in a mess and very hard to recognize, hence the desired reconstruction result can not be reached. But the CNN depth estimation from single image can improve this situation. But still not enough to get a acceptable result. Also, the speed of this combination is too slow, with a GTX 1060 GPU,and CUDA 8, tensorflow 1.0, this can only run with about 4 FPS. This maybe not enough to work with real time SLAM.

So there will be more work can be done to further improve this algorithm, such as, CNN semantic segmentation in input image, because we can see that most noise in the point cloud is from "nothing", so if we can make segmentation and only input the object we need, there

may be less noise. Also we can use CNN to operate point cloud segmentation as a post-process to further reduce noise. There has been some work about this.

And last but most important, to increase the network speed, to improve the SLAM performance to meet the requirement in practice.

Bibliography

[1] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[2] Jakob Engel, Jurgen Sturm, and Daniel Cremers. Semi-dense visual odometry for a monocular camera. In *Proceedings of the IEEE international conference on computer vision*, pages 1449–1456, 2013.

[3] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *CVPR*, volume 2, page 7, 2017.

[4] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

[5] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, June 2014.

[6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based

learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[7] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[8] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[9] JÃ$\frac{1}{4}$rgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015.

[10] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 373–374, New York, NY, USA, 2014. ACM.

[11] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2017.

[12] Wen-tau Yih, Kristina Toutanova, John C. Platt, and Christopher Meek. Learning discriminative projections for text similarity measures. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, CoNLL '11, pages 247–256, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.