Project Report - Dense 3D Reconstruction for SfM

LIU Meng, WU Kai

Robotics, Cognition, Intelligence - Technische Universität München

Abstract

The exercise phase has shown that the real world scene can be reconstructed with features, so called sparse reconstruction. Our project explores another way to digitize the world, namely dense reconstruction using disparity maps. To do so, we seek help from different libraries and applications to generate high quality depth map of image pair, e.g. OpenCV, Libelas, image_undistort and PSMNet, etc., then align them with Voxblox. Image_undistort gives the best reconstruction result then followed by our implementation with OpenCV. The code of our project is available at GitLab on branch `deepslam`, `https://gitlab.vision.in.tum.de/visnav_ss19/s0013/visnav/tree/deepslam/project`

Contents

1 Introduction

In this project, we intend to use existing libraries to construct a feasible pipeline to reconstruct the dense 3D scene from the EuRoC MAV dataset [2] based on the result from structure from motion (SfM). As part of the homework in this practical course, we have managed to select keyframes among stereo images in the dataset and track the trajectory of the stereo camera. For the reconstruction purpose, depth images are to be calculated from pairs of stereo images. These depth images are represented by pointclouds and can then be fused according to their corresponding camera poses.

2 Related works

Structure from motion (SfM) constructs the 3D scene using all images captured by single camera or stereos. Its core idea is to determine camera trajectory and landmark positions in the world coordinate by finding correspondence points in several images with algorithms such as bundle adjustment (BA). In contrast to the sparse map built by SfM in the exercise phase, dense maps are much more appealing in the sense of digitize the world, like cultural heritage, through accurate 3D model. Therefore, depth estimation on the entire frame is needed to preserve the information. As the result, a dense reconstruction can be acquired with depth expressed in camera frames and the camera positions of each frame.

2.1 Epipolar geometry

As one of the fundamental elements of the multiple views reconstruction, stereo processing largely influences the quality of final model. A typical path starts from image rectification given a calibrated binocular stereo. The epipolar geometry is applied to achieve perfect co-planarity between the image pair. Having the images rectified makes searching the correspondences in depth estimation much more practical, as block matching along the horizontal line is linear runtime to the size of frame.

Considering the efficiency and quality, variations of block matching are proposed in papers, e.g. Geiger et al. [5] and Hirschmuller [6], and also implementations involve convolutional neural networks (CNNs), e.g., Chang et al. [3]. The basic idea of block matching is shown in the figure 1. The horizontal shift of the same point shared in calibrated and rectified stereo represents the disparity of the current point. Indicating points locate in green box and red box in the figure. The closer the object to the camera stays, the larger is the disparity between the scenes.

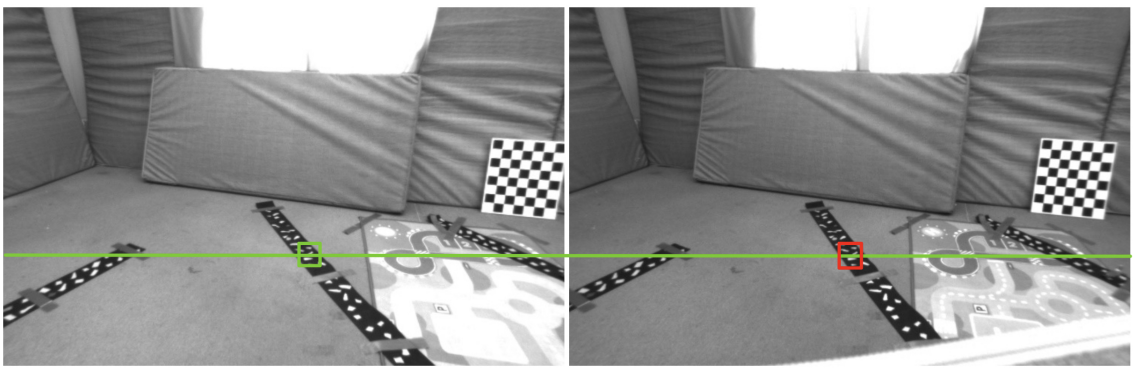

Figure 1: Basic block matching method illustration, searching correspondences along the horizontal line, the differences of positions represent the disparity

2.2 Map representation

Another essential problem for the reconstruction to consider is to represent the map, i.e. the scene, in a proper way. According to different applications, robotic mapping can be divided into two categories: maps that a robot can use for planning and high fidelity 3D reconstruction for human use. While in planning applications lowest resolution representation is preferred to use that can still describe the environment and free space relative to the robot size, mapping for reconstruction requires high resolutions, small feature sizes and color[8].

For the reconstruction purpose, a common representation of the space is the signed distance function(SDF), where surface interfaces are represented as zeros, free space as positive values that increase with distance from nearest surface and occupied space with similarly defined negative values. In practice truncation of the SDF is required to avoid surfaces interfering[4].

Euclidean Signed Distance Fields(ESDF) are used to represent the space for planning which provides distance to the nearest obstacle and also collision gradient information.

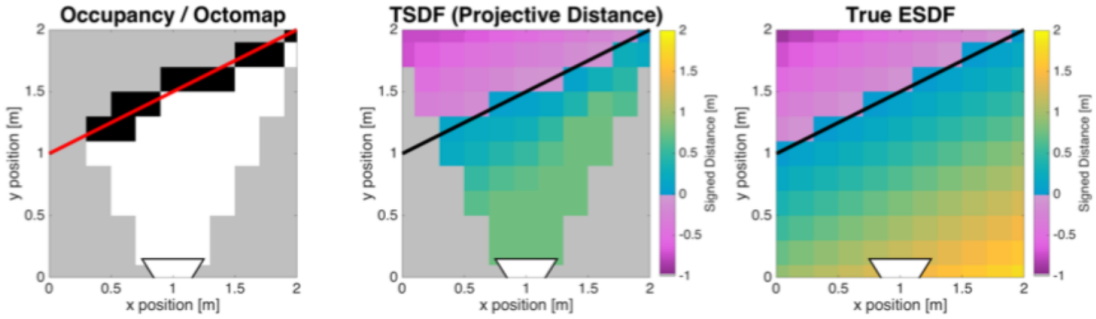

Figure 2: Different representations of the space[8]

Truncated Signed Distance Fields(TSDF) has been widely used for 3D reconstruction as they allow high-resolution mapping in real-time on GPU[7]. It has been further adapted to CPU platform by enabling it to handle large voxel sizes[8], and the author has implemented a handy ROS library – Voxblox – to calculate TSDF for reconstruction, ESDF for planning and mesh for visualization(see Figure 3). Therefore after generating depth map from stereo images, the reconstruction part of this project will mainly utilize functions in this library.

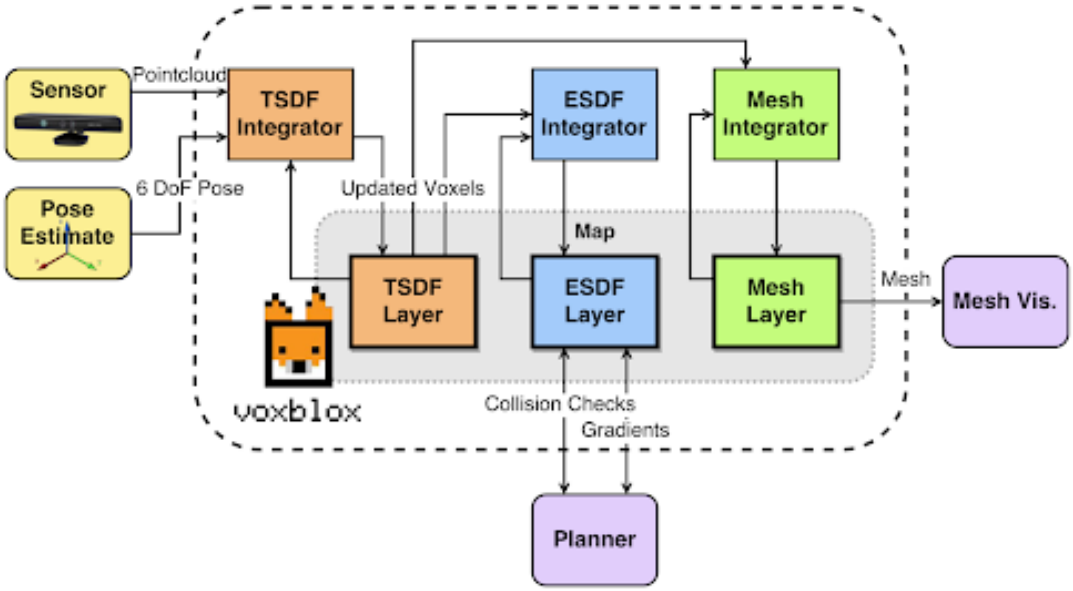

Figure 3: Voxblox framework[8]

3 Pipeline

Our reconstruction pipeline mainly consists of two parts, namely stereo processing and reconstruction with Voxblox, see the figure 4. The highlight parts are our main concerns. The implementation takes the camera poses which are pre-calculated in SfM exercises and depth information computed in stereo processing as input. A 3D room as output is reconstructed with VoxBlox.

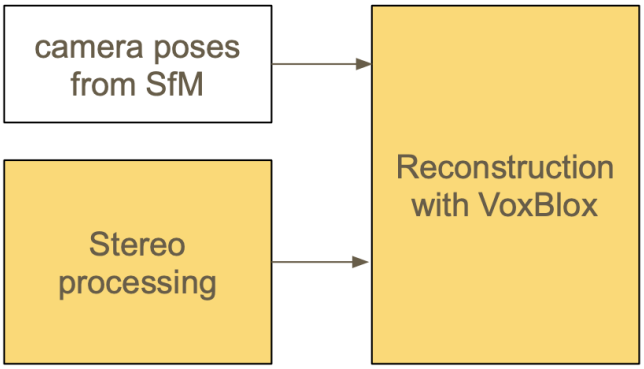

Figure 4: Pipeline

3.1 Stereo processing

The stereo processing part starts from given a pair of images and extends until the depth of each scene is stored in the point cloud file. As shown in the figure 5, the whole process can be done through OpenCV, or another library called ethz-asl/image_undistort which partly built on OpenCV. Since the later can be fully embedded in Voxblox, we will not discuss it exhaustively in this section. Furthermore, there exist two more methods to

compute disparity maps, which is the core part of stereo processing and the focal point here. In the following part of this section, we will break down this figure into details.

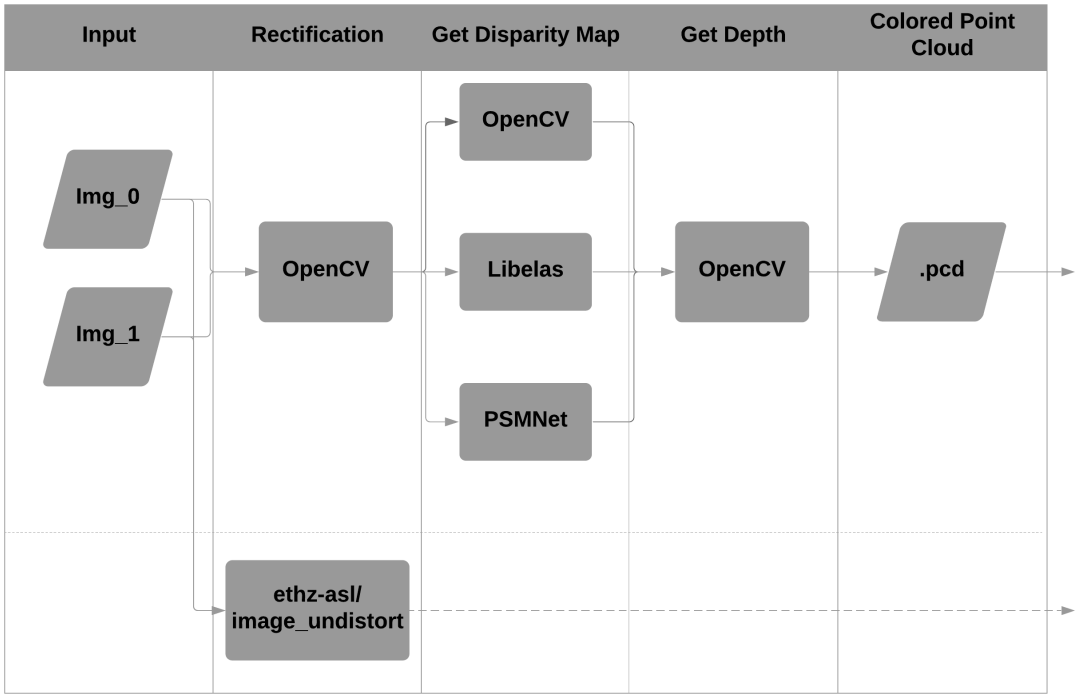

Figure 5: Libraries used during the process of generate depth maps from image pairs

As we mentioned before, the entire first step of our project can be done only using OpenCV, the following code 1 shows an overview. For every rectified images a disparity map is computed, then followed by a weighted least square filtering. More information can be found in our code and in sub-section Disparity map generation.

```
for imgid in os.listdir(img_path_0):
    imgL = cv2.imread(img_path_0 + imgid, -1)
    imgR = cv2.imread(img_path_1 + imgid, -1)
    disp0, disp1 = computeDispMap(imgL,imgR, map0x, map0y, map1x, map1y,
    left_matcher, imgid)
    disp0 = np.int16(disp0)
    disp1 = np.int16(disp1)
    filteredImg = wls_filter.filter(disp0, imgL, None, disp0)
    filteredImg = cv2.normalize(src=filteredImg, dst=filteredImg, beta=0,
    alpha=255, norm_type=cv2.NORM_MINMAX)
    filteredImg = np.uint8(filteredImg)
    cv2.imwrite(os.path.join('./results/disparity_map_opencv', imgid),
    filteredImg)
    break
```

Listing 1: OpenCV implementation of generation disparity map and post-processing

1. **Stereo images rectification**
   At this step, inputs are raw left eye and right eye images directly taken from stereo cameras, and the outputs are rectified images which means the corresponding points shared by two eyes should lie on the same horizontal line. The OpenCV

function `stereoRectify` is called here. Since we have potentially hundreds of image pairs to process, wrapping same sub-steps can be much more efficient. Thus, `cv2.initUndistortRectifyMap` is used to compute an undistortion and rectification transformation map for each eye, and for individual image, function `cv2.remap` can apply a geometrical transformation to get the rectified image. The efficiency is built on the assumption that the distortions of the stereo camera don't change over time.

```
R1, R2, P1, P2, Q, roi1, roi2 = cv2.stereoRectify(
    cameraMatrix1=cameraMatrix0,
    distCoeffs1=distCoeff0,
    cameraMatrix2=cameraMatrix1,
    distCoeffs2=distCoeff1,
    imageSize=(752, 480),
    R=R,
    T=T#, alpha = 1
    )

```

Listing 2: rectification

2. **Disparity map generation**
   After we get the rectified images, we can easily compute the disparity. Since this is an experimental project, we have tried several methods, e.g. deep neural network, libraries and software. All of them have the common inputs, namely the rectified image pairs. For more information about the different trials, please check section 4. In this section, only the methods which have generated meaningful results during our experiment will be described here.

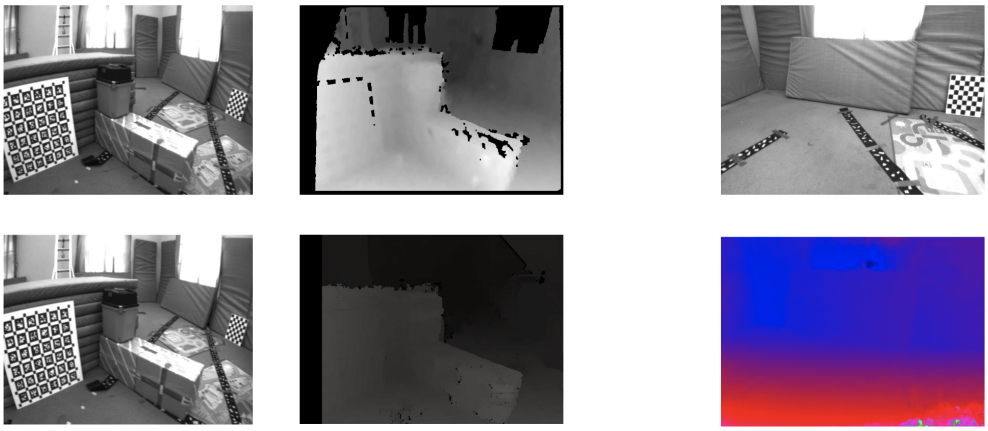

Figure 6: Disparity maps generated through different methods, left two are the original image pair

OpenCV We choose the semi-global block matching algorithm `cv2.StereoSGBM_create` to compute the disparity map. The parameters are set as follows.

```
stereo = cv2.StereoSGBM_create(
    numDisparities=64,  blockSize=3,
    minDisparity = 0,
    # SADWindowSize = 11,
    P1 = 120,
    P2 = 240,
```

```
    disp12MaxDiff = -1,
    preFilterCap = 31,
    uniquenessRatio = 0,
    speckleWindowSize = 500,
    speckleRange = 3)

```

Listing 3: Compute disparity map in OpenCV

The parameters are mainly chosen according to the documentation [1]. Among them, the `blockSize` has the greatest influence on the result quality. Too small block size will result in higher noise level on the frame, while too large block size can lose some subtle but important details.

Even though the parameters are set as the most suitable for the dataset, the disparity maps still contain matching errors. Therefore, a post-processing is needed to rule out the artifacts brought by pursuing efficiency, namely the trade-off between quality and computational speed. As discussed in [1], uniform texture-less areas, half-occlusions and regions near depth discontinuities are the sources of the mentioned problem, and the enhancement method `cv2.ximgproc.DisparityWLSFilter` is also proposed in [1]. The figure at the middle bottom of 6 shows the final result.

Libelas This is another free software [5] that robustly computes disparity map by adding a prior on the disparities. We found it during the research period of the project. Due to the limited time, we only did slightly modification on the original released version of this software to make it work on the EuRoC Vicon Room dataset [2]. The figure at the middle top of 6 shows the final result.

PSMNet Deep neural network is also a trend for disparity map generation. PSMNet stands for Pyramid Stereo Matching Network [3], and one of its pre-trained models was at the top rank on the KITTI dataset leader board when the authors proposed this method. It was still top 10 when our project started. Unfortunately, due to hardness of setup the running environment, we only get one reasonable result shown at the right bottom of figure 6. The original image of this disparity map is at the top right. More detail will be discussed in section 4.

3. **Point cloud generation**
   To compute the real depth value, a OpenCV function `cv2.reprojectImageTo3D` is called here.

3.2 Dense reconstruction

So far we have obtained pointcloud of each pair of stereo images and its corresponding camera pose from SfM which are all we need to reconstruct the scene.

The Voxblox package is built on ROS. The voxblox node takes ROS messages of pointcloud and its camera pose as input. By carefully assigning the messages with correct timestamps and frame coordinates(publisher node see file pcl2msg_node.cpp), the voxblox node can fuse pointclouds of given keyframes and generate the mesh representation for it(see Figure 6).

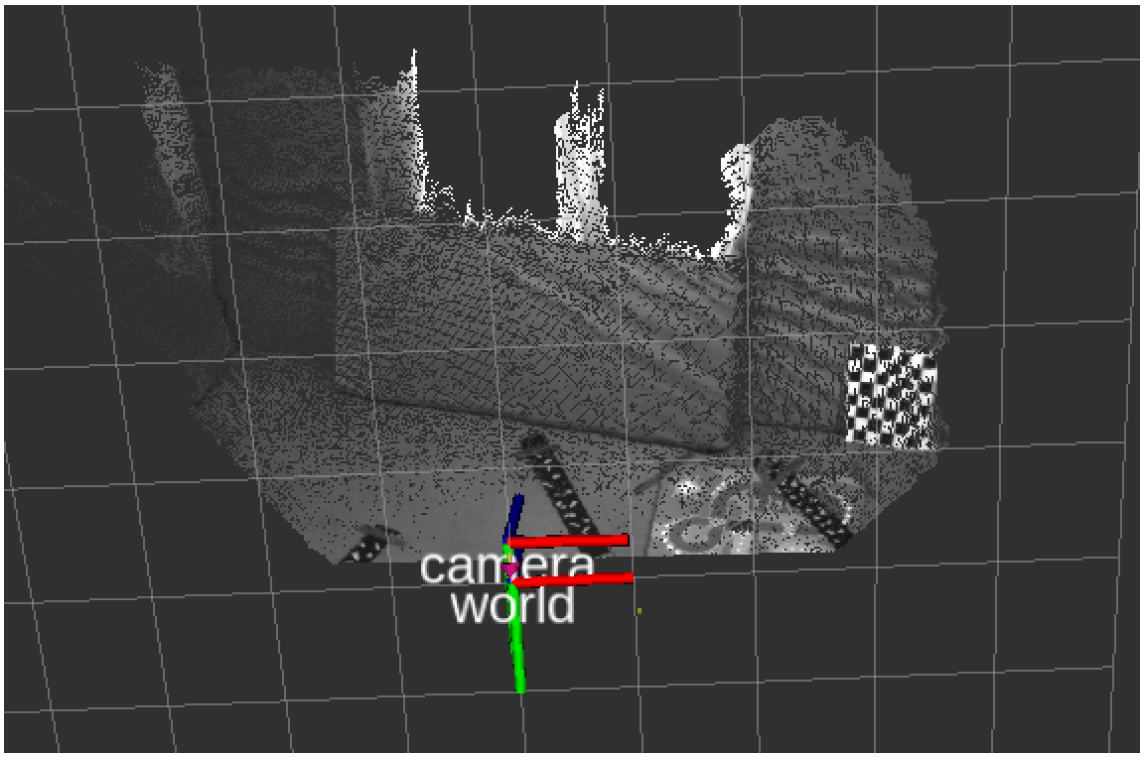

Figure 7: Point cloud shows one scene from the dataset

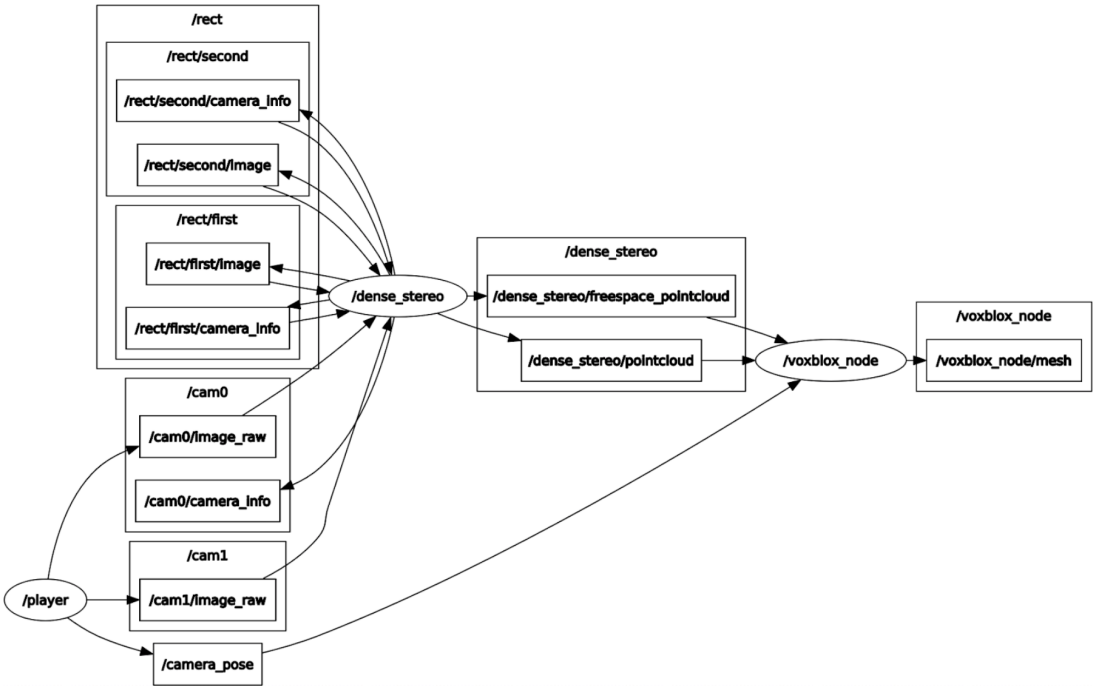

Figure 8: Workflow of ROS nodes

3.3 Post-processing of depth maps

In section 4 one can see that the reconstruction using depth images directly from OpenCV functions has odd performance at boundaries, windows and there are holes in the floor. To improve it, a common way is to execute a post-processing for the depth images.

The authors of Voxblox provide another ROS package "image_undistort" for stereo images processing. In this package, same OpenCV functions as used in section 3.1 are called to calculate depth image from stereo images. In addition, several manually designed criteria are used to handle invalid and abnormal depth values, so that holes on planar surfaces can be filled, invalid depth values are substituted with valid ones according to their neighborhood, etc.

As show in Figure 6, the dense_stereo node takes a pair of raw stereo images as input, it rectifies the images, calculates the depth image and outputs pointcloud for each pair of stereo images. Then same reconstruction procedure is executed by the voxblox node.

4 Results

During the exploration, different methods show different characters when solving the same tasks. As shown in the table 1, we summarize them into four categories and compare with each other in a qualitative manner due to the limitation of time.

Library	Quality	Input format	Hardware	Flexibility
OpenCV	Middle	-	-	Good
Libelas	Bad	.pgm	-	Middle
PSMNet	Good	.jpeg/.png	GPU	Bad
ethz-asl/image_undistort	Good	ROS msg	-	Middle

Table 1: Qualitative comparison between different methods for generating depth maps, - stands for "no (clear) requirement"

In general, OpenCV is well preformed in whole process, it can not only provide complete solution from image rectification to depth map generation, but also offer acceptable quality of result. Another compact package which offers full solution is `ethz-asl/image_undistort`, which has not a mature documentation to look up. Comparing the mentioned two methods, even though the later depends partly on the former, it generates better results. In stead of using OpenCV method continuously, the package `ethz-asl/image_undistort` has its own way to post process the vanilla disparity map. It replaces invalid disparities like where represent the position of the sky but contain too large values. Theoretically, it can compute a disparity map without holes, and which means an extremely dense reconstruction in the future.

Libelas is also a compact command-line software for computing disparity maps. For simplicity, we only consider the released version here. It neglects the holes generated from texture-less surfaces. More importantly, it has texture copying effect where there has no protrusions and indentations. This phenomenon can be shown on the camera calibration board in figure 6. One inconvenience brought by this software is that it only accepts `.pgm` format, which means an additional format conversion step.

Our experiments with PSMNet started at the very beginning, and it also lasted longest. However, comparing to the effort we have paid the result is rather frustrated. The biggest problem is to set up the suitable environment to run the pre-tained model of this network. The whole discussion can be viewed in the GitHub issue `Test the pre-trained model`

`with EuRoc Vicon Room #138`. According to the author, it requires `Python2.7`, `PyTorch` with version later than `0.4.0`, `torchvision 0.2.0`, and a GPU with 6GB to 8GB of memory. Therefore, we chosen the Google Colaboratory with configurations shown below 4.

```
('Gen RAM Free: 12.7 GB', ' | Proc size: 230.5 MB')
GPU RAM Free: 11441MB | Used: 0MB | Util   0% | Total 11441MB
```

Listing 4: Google Colaboratory configurations

Nevertheless, we can not get the disparity map which has a similar quality with the one provided by the author using our dataset (shown at the bottom right of figure 6).

Due to the problems described above, we use only depth images from OpenCV and image_undistort libraries for the reconstruction experiment. The left image in Figure 9 is the scene that is reconstructed with OpenCV depth images of key frames and camera poses from SfM. In the right, same camera poses are used, but depth images of key frames are generated by image_undistort package. In Figure 10 is the reconstruction using depth images of 2912 pairs of stereo images(generated by image_undistort package) and 14629 messages of ground-truth camera trajectory measured by VICON(6D motion capture system).

One can see that with our chosen 82 key frames, the reconstruction of the scene can is already relatively satisfying compared with the reconstruction from 2912 frame in the right image.

After post-processing the depth images, large holes on the surface of floor and the box in the middle are filled and the abnormal depth values near the window caused by the extremely large intensity values are discarded. These manually designed criteria in the post-processing however leaves holes in the position of windows. Therefore we could expect that instead of a manually designed hard-coded way to refine depth images, with a more powerful depth image generator like neural networks, the quality of reconstruction can be further improved.

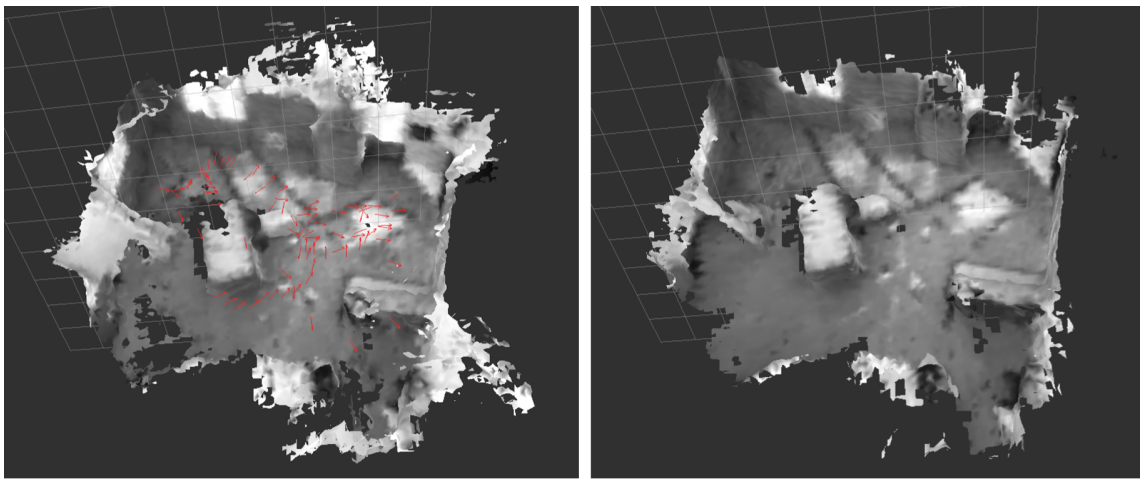

Figure 9: Reconstruction results with key frames and camera poses from SfM

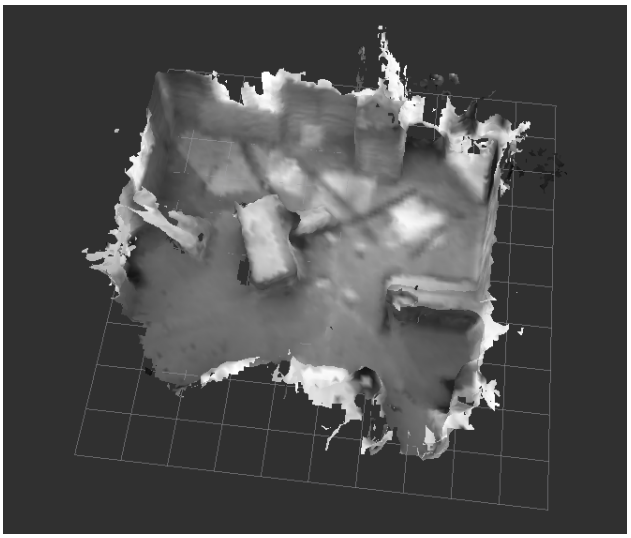

Figure 10: Reconstruction results with grund-truth

5 Conclusion

From the perspective of the purpose of this project, the pipeline works. With a portion of less than 3% chosen key frames among all images, a satisfyingly dense map can be already generated and the post-processing of depth maps to discard invalid depth values and fill holes can further improve the quality of the map.

However, in the sense of efficiency, it definitely needs some adjustments. First of all, implementations incorporating with different libraries, software or even environments are painful. It requires lots of time to deal with the inconsistency brought by crossing the interfaces, including the file formats, data structures, and hardware requirements. For example, the stereo processing part is written in python, while the reconstruction part is based on ROS. Thus one needs to tediously convert all input data needed by Voxblox into corresponding ROS messages. Therefore, in the future work, how to bring all the steps in one frame and preserve the convenience of the modularity is the main topic.

In general, for disparity map generation, traditional methods like OpenCV and image_undistort are acceptable. Whether the neural network methods can re-produce a higher quality result with different datasets as their commitments needs more researches.

References

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016.

[3] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5410–5418, 2018.

[4] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. 1996.

[5] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In *Asian Conference on Computer Vision*, Queenstown, New Zealand, November 2010.

[6] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, Feb 2008.

[7] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *ISMAR*, volume 11, pages 127–136, 2011.

[8] Helen Oleynikova, Zachary Taylor, Marius Fehr, Juan Nieto, and Roland Siegwart. Voxblox: Building 3d signed distance fields for planning. *arXiv*, pages arXiv–1611, 2016.