3D SENSING AND MAPPING USING MOBILE COLOR AND DEPTH SENSORS

BY

RAMANPREET SINGH PAHWAA

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Doctoral Committee:

Professor Minh N. Do, Chair
Associate Professor Derek Hoiem
Professor Mark Allan Hasegawa-Johnson
Senior Research Scientist Dr. Jiangbo Lu
ABSTRACT

An important recent development in the visual information acquisition field is the emergence of low cost depth sensors that measure the scalar distance between the camera and the surrounding objects present in the scene. These depth sensors project infrared rays that are invisible to humans to measure the scene distance at each pixel. These sensors already have had a significant impact on various fields such as computer vision, gaming, augmented reality, and robotic vision. However, like every new technology, depth cameras also suffer from severe limitations such as low resolution, significant noise, lens distortion, and inability to work in outdoor environments. Depth cameras need to be calibrated accurately before they can be used along with color cameras to perform various tasks such as 3D reconstruction, action recognition, scene sensing and augmented virtual reality.

This thesis investigates novel methods to measure and correct for these distortions and use the denoised measurements for various applications in vision related fields. In particular, we tackle the following problems:

First, we propose a novel algorithm that takes in few depth images and utilizes them to simultaneously denoise and calibrate time-of-flight based depth cameras. Our formulation is based on two key elements. We first use depth planarization in 3D to denoise the depth at each corner pixel. Thereafter, we use these improved depth measurements along with the corner pixel information to estimate the calibration parameters using a non-linear estimation algorithm. We demonstrate that our framework estimates the intrinsic and extrinsic calibration parameters more accurately using fewer images and corners than are needed for traditional camera calibration. We evaluate our approach on both a synthetic dataset where ground truth information is available, and real data taken from a photon mixing device (PMD) camera. In both cases, we demonstrate that our proposed framework outperforms traditional calibration technique without significant increase in computational complexity.

Second, we use the depth information provided by such cameras along with
the color information for 3D object proposals in a given scene. We use a generic 2D object proposal technique as an input to our system and perform depth based filtering to create a heatmap of each frame by exploiting the scene geometry. We further use these heatmaps to remove any supporting planes present in the scene. Thereafter, we fuse the heatmaps of each frame in 3D using camera pose to build a 3D point cloud of the scene and assign each point a ranking based on its importance in the scene. We perform density based clustering on these top ranked points to compute precise 3D bounding boxes in the scene that have a high probability of containing an object of interest.

Third, we integrate depth sensors and external geometry of the scene to robustly stitch images captured in a cylindrical tunnel where the camera moves forward in a spiral fashion. We utilize structure-from-motion (SfM) to estimate camera pose between adjacent frames. We exploit scene geometry to identify outliers among matching points and use bundle adjustment (BA) to improve the camera pose. We use depth sensors attached to the color camera to estimate the camera’s translation. Thereafter, we create an immersive 3D display in Unity 3D rendering engine to display the stitched scenes in a cylindrical projection where the user can fly through the scene using keyboard and mouse controls.

In the future, we intend to improve bundle adjustment for automatic stitching of tunnel-like scenes by exploiting the known geometry of the scene to make it more robust to outliers.
To my family, for their unconditional love and support.
I owe my deepest gratitude to my adviser Prof. Minh N. Do for providing me with an invaluable opportunity to work on state-of-the-art depth camera technology under him. He has always been approachable and encouraging and has provided invaluable insight for my research work. I am also deeply indebted to Dr. Tian Tsong Ng whose encouragement, guidance, and support from the initial to the final level enabled me to develop an understanding of the subject. I would also like to thank my colleagues and research collaborators for their generous assistance and beneficial discussions, especially Dr. Jiangbo Lu, Dr. Nianjuan Jiang, and Dr. Vijay Chandrasekhar. I would especially like to thank my parents, Mr. Nirmal Singh Pahwa and Mrs. Sarabjit Kaur, and my brother, Mr. Taranpreet Singh Pahwa, for their never-ending support, love, affection and faith in me. Lastly, I offer my regards to all of those who supported me in any respect during the completion of my work.
# TABLE OF CONTENTS

LIST OF TABLES ................................................. viii

LIST OF FIGURES .............................................. ix

CHAPTER 1 INTRODUCTION ................................. 1
  1.1 Motivations ........................................... 1
  1.2 Problem Statement .................................... 4
  1.3 Thesis Contributions and Summary .................. 5

CHAPTER 2 CALIBRATION OF DEPTH CAMERAS ............. 6
  2.1 Introduction ........................................... 6
  2.2 Calibration Parameters ............................... 7
  2.3 Color Camera Calibration ............................. 9
  2.4 Depth Camera Calibration ............................ 22
  2.5 Experimental Results ................................. 25
  2.6 Conclusion ........................................... 27

CHAPTER 3 LOCATING 3D OBJECT PROPOSALS: A DEPTH-
  BASED ONLINE APPROACH ............................... 29
  3.1 Introduction ........................................... 29
  3.2 Related Work .......................................... 31
  3.3 Proposed Approach ................................... 33
  3.4 Depth-Based 2D Object Proposal Filtering .......... 35
  3.5 Multi-view Fusion and Refinement for a Global 3D Heatmap 40
  3.6 Experimental Evaluation ............................. 50
  3.7 Summary ............................................. 60

CHAPTER 4 SPIRAL STITCHING OF TUNNEL IMAGES ........... 62
  4.1 Introduction ........................................... 62
  4.2 Related Work .......................................... 64
  4.3 Cylindrical Projection ............................... 67
  4.4 Our Framework ....................................... 69
  4.5 Our Approach ........................................ 73
  4.6 Experimental Results ............................... 81
  4.7 Conclusion ........................................... 91
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Avg. 3D error between groundtruth corners and projected corners in mm.</td>
</tr>
<tr>
<td>3.1</td>
<td>Comparison between edge-boxes and our technique per frame on UW-RGBD Scene dataset. SR and DR refer to success rate and detection rate respectively.</td>
</tr>
<tr>
<td>3.2</td>
<td>Analysis of average run-time performance per frame of our approach. Our experiments were conducted on a single core Intel Xeon E5-1620 CPU.</td>
</tr>
<tr>
<td>3.3</td>
<td>Comparison of average IoU for various objects present in UW-RGBD dataset. The benefits of each of our contributions - background suppression, rejection of odd sized bounding boxes, and supporting plane removal are clearly visible.</td>
</tr>
<tr>
<td>3.4</td>
<td>Analysis of our 3D object proposals on UW-RGBD dataset in comparison to [1], [2], and [3]. We achieve near runtime performance if we downsample the data for our analysis. Our experiments were conducted on a single core Intel Xeon E5-1620 CPU.</td>
</tr>
<tr>
<td>4.1</td>
<td>The reprojection error reduces significantly after perform three-step pruning on baseline RepMatch.</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Examples of depth sensors</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Image formation using projective geometry</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>A scene captured by (a) depth sensor and (b) color sensor</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>The two types of distortions present in an imaging system are radial and tangential distortion. Radial distortion in (a) caused the rays farther from the optical center to bend more than the rays nearer to the optical center [4]. Tangential distortion in (b) happens due to manufacturing defects where the imaging plane of the camera is not perfectly parallel to the lens [4]</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>A sample color and depth image captured by PMD camcube for the calibration process. The checkerboard is moved around the scene and captured using a depth camera</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>The checkerboard is transformed per frame from world coordinate frame to camera coordinate frame using the extrinsic camera parameters. Then it is projected onto camera plane using intrinsic camera parameters</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>A modular approach for standard camera calibration</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>We use the interior corner along with intensity information to extract the white checkerboard regions. These regions do not suffer from Poisson noise and thus can be used to estimate the 3D checkerboard plane</td>
<td>24</td>
</tr>
<tr>
<td>2.6</td>
<td>Checkerboards projected in 3D using (a) Original depth information (b) 3D planarization. The 3D planarization helps in removing the Poisson noise present in darker regions of the checkerboard making the calibration process more accurate</td>
<td>24</td>
</tr>
<tr>
<td>2.7</td>
<td>Relative error in focal length for noisy synthetic data. We magnified the scale of the vertical axis in the cases of 9-36 corners to highlight the accuracy of our calibration scheme</td>
<td>26</td>
</tr>
<tr>
<td>2.8</td>
<td>Relative error in focal length for PMD depth camera</td>
<td>27</td>
</tr>
</tbody>
</table>
3.1 Scene data is captured using an RGB-D camera (best viewed in color). The aligned color and depth images are used for camera pose estimation (illustrated by the red trajectory). Our approach uses this information along with any generic 2D object proposals to fuse and filter the data in 3D and output precise 3D object proposals (denoted by colored 3D bounding-boxes).

3.2 Our proposed framework for 3D object proposals.

3.3 Top ranked object proposals are displayed for images using Non-Maximum Suppression (best viewed in color). Object proposals are displayed in order of their confidence. Object proposal with highest confidence is displayed first. Any object proposals that share more than 10% region, i.e. intersection over union (IoU) greater than 0.1, are not shown. Using a simple NMS approach fails to select the best object proposals.

3.4 (a) Color image. (d) Corresponding depth image. (b) Baseline heatmap, $\hat{H}$. (e) Our heatmap, $H_{2D}$. (c) Baseline heatmap overlaid on color image. (f) Our heatmap overlaid on color image. We show a comparison between the baseline heatmap $\hat{H}$, and our improved heatmap $H_{2D}$. Our depth-assisted heatmap is able to filter out the background from the objects of interest using depth based statistics for every object proposal.

3.5 Depth based filtering of 2D object proposals (best viewed in color). We only display a handful of object proposals for clarity purposes. In the soft filtering process, the blue and magenta bounding boxes are detected to contain background and undergo depth based background suppression while the red and green bounding boxes are detected to contain no major background. After background masking process, we check the actual size of each bounding box using depth information. The red and magenta bounding boxes are identified as areas that are either too small or too big in size or do not contain depth information (holes in depth image). These 2D object proposals are discarded. The blue regions, after background suppression, and green regions are detected to potentially contain good information and are accepted as valid object proposals.

3.6 Our plane removal process takes into account the heatmap of the scene. A sample 3D point cloud and its weighted heatmap are shown in (a) and (b) respectively. We display the most dominant planes in order of number of inliers in (c)-(f). The plane in (f) is selected for plane removal as its heatmap based confidence is highest among top ranked planes.
3.7 (a) Color image. (b) Matched pixels. (c) Weighted 2D heatmap, $H_{2D}$. (d) Heatmap after plane removal, $\tilde{H}_{2D}$. (e) Global heatmap, $H_{3D}$ projected onto image plane. Left column displays different RGB frames from table_1 scene in [5]. 2nd columns shows the matched pixels of current frame with the previous frame. 3rd column highlights our weighted 2D heatmap after depth based filtering. 4th column displays our refined heatmap after plane removal. Finally, the 5th column shows the current global heatmap of the entire scene. The global heatmap has been scaled for visualization.

3.8 We show our 3D heatmap generation process using image warping. We initialize our global heatmap, $H_{3D}$ after first frame. Once we obtain the current frame, we warp the previous image on top of the current image and find the matching points. The matching points are shown in various unique colors and the unmatched points are shown in gray. We consolidate these points’ confidence value by adding their individual confidence. We update their 3D location in world coordinate frame and color information by averaging their individual measurements. The unmatched points (shown in grey) are added to the global heatmap with their respective confidence values and unit frequency.

3.9 Once we obtain the top ranked points, we perform a final plane removal step to remove any leftover supporting plane points. We perform density-based spatial clustering to find the most dominant point clusters and estimate their bounding boxes correcting for direction of gravity.

3.10 Final results for a sample scene in UW-RGBD dataset for our approach after removing one component. We obtain results in (a) without depth based filtering as described in Section 3.4. Results in (b) and (c) are obtained when background masking and bounding box rejection are not performed respectively. (d) shows results without plane removal. (e) shows our result when all of these components - background masking, bounding box rejection, and plane removal - are used together.

3.11 Our 2D and 3D recall rate with varying threshold for IoU on UW-RGBD dataset. Our 2D proposals comfortably outperform [3]. Since we exploit scene geometry, downsampling images by 2 does not have a significant impact on our recall rate.

3.12 Our 3D object proposals for UW-RGBD dataset (best seen in color). Left column shows the point cloud of the different scenes in UW-RGBD dataset. 2nd column shows our top ranked filtered points and finally 3rd column highlights the resulting 3D object proposals based on density based clustering.
3.13 We compare Precision-Recall measurements for four objects in RGBD scenes dataset. We projected our 3D bounding boxes onto image plane to obtain our 2D bounding boxes and used the groundtruth bounding boxes to obtain these results. 

3.14 Left column shows the point cloud of various scenes in our dataset. 2nd column represents our top ranked filtered points and 3rd column shows the resulting 3D object proposals based on density based filtering. Our approach can locate and cluster even thin objects such as books. However, due to flying-pixel depth noise, sometimes two extremely close objects are clustered together.

4.1 (a) Deep tunnel sewers [6]. (b) View from inside the tunnel [6]. (c) Our custom UAV design. (d) The UAV descends into the tunnel from one access shaft, flies through the tunnel and ascends to the surface from the next access shaft.

4.2 The UAV is assumed to be moving horizontally during the capturing process. This constant horizontal motion of the UAV coupled with the rotating camera results in a panoramic spiral image.

4.3 Cylindrical projection - A 3D point is projected onto a cylinder with unit radius to obtain cylindrical coordinates. These cylindrical coordinates are then eventually flattened out onto a planar image.

4.4 (a) A sample brick image. (b) Image obtained after cylindrical projection.

4.5 We triangulate the initial position of the UAV using the depth sensors located on the 4 sides of the UAV. We ignore the depth measurement of the sensor pointing downwards as presence of liquids at the bottom of tunnel results in an inaccurate measurement.

4.6 (a) The UAV is allowed to move a maximum distance of $r_1$ mm from the center of the tunnel. (b) As this distance from center of tunnel increases, the maximum allowable distance moved by the UAV decreases to allow some overlap in between two images with the same camera rotation and ensure no dead-space during the offline stitching process.
4.7 (a) SURF matching points after outlier rejection when camera rotates 36°. (b) RepMatch matching points when camera rotates 36°. (c) SURF matching points after outlier rejection when camera moves forward. (d) RepMatch matching points after outlier rejection when camera moves forward. SURF based matching struggles in presence of repeated structures such as floor tiles while RepMatch gives us more dense accurate point matching which results in a more reliable camera pose.

4.8 We show a few examples of RepMatch [7] point matching. In (a) and (b) RepMatch correctly identifies matching point giving us a good camera pose estimate. However, in scenes represented by (c) where objects such as the trolley carrying the camera move along with the camera, RepMatch incorrectly identifies points lying on these as matching points resulting in a noisy camera pose. We avoid these wrong matches by using a manual mask.

4.9 (a) Matching between rotationally consecutive frames. (b) Matching between both rotationally consecutive frames and horizontally moving forward frames. We add links between horizontally moving forward frames (when camera completes a full 360° rotation). This allows for pose estimation between these additional frames making the bundle adjustment step more robust.

4.10 (a) Average 2D reprojection error. (b) Average 3D error in mm. The camera rotates 36° per frame and completes full 360° rotation after ten frames. Introducing the vertical links between every i and i + 10 frames significantly lowers the 2D reprojection and 3D error during the Bundle Adjustment process.

4.11 Top row shows the unity simulation when camera is rotated by 0° and 180° with respect to y axis (green arrow). Bottom row displays the images captured by the virtual camera at these two angles respectively.

4.12 Top row displays the images captured from unity when the camera is rotated from 0 to 360 degrees. Bottom row displays the stitched image obtained after our stitching process.

4.13 (a) Stitched cylindrical image without accounting for camera location off center. (b) Stitched cylindrical image after accounting for camera’s location off center.
4.14 Stitched cylindrical image when the camera moves forward by (a) 18 cm per rotation. (b) 20 cm per rotation. We only stitch the $0^\circ$ and $180^\circ$ images to highlight the gaps between views. As the maximum allowable speed of the camera is 18 cm per rotation, we start observing holes in the cylindrical image if the camera’s speed increases above the maximum allowable speed.

4.15 We add random jitter and movement to the camera after each image capture. Using baseline rotation and translation leads to an extremely inaccurate and incoherent stitch. Using SfM and BA enables us to improve the camera pose per image pair and results in a much more accurate and coherent cylindrical projection of the scene.

4.16 (a) Underpass used to capture our dataset. (b) Our camera setup. (c) Our rotating mechanism. (d) Our camera and depth sensor capture data simultaneously and save it on an external storage device. We manually move the tripod to capture the data while the camera and depth sensor rotate automatically during this process.

4.17 We perform a stitch of the underpass using RepMatch [7]. The camera rotates full $360^\circ$ before it is manually moved forward. The top row displays the cylindrical stitch of the underpass and the scene rendered in a rectangular room in Unity.

4.18 (a) 3D reconstruction of floor points after 60 frames without removal of wrongly matched points. (b) 3D reconstruction of floor points after 120 frames after removal of wrongly matched points. (c) 3D reconstruction of floor points after 230 frames after removal of wrongly matched points. The floor points still roughly lie on a plane after the camera moves 22 m forward demonstrating the accuracy and robustness of our SfM and BA process.

4.19 We perform a stitch of the underpass using RepMatch [7]. The camera rotates $36^\circ$ and manually moved forward per frame. We display the left and right walls after cylindrical stitching.

4.20 A hollow cylinder is created in Blender and UV mapped. This cylinder is imported in unity to create both straight and curved tunnels based on the user requirements. The user can move freely inside the tunnel and move closer to the boundaries to carefully inspect for any damage in the tunnel.

4.21 The “Cylindrical” dataset is rendered in Unity. We add front and end walls to signify start and end of the tunnel passage.

A.1 PMD camcube’s smart pixel as described in [8].

A.2 The cross correlation function for a single modulation period for PMD camera [8].
CHAPTER 1

INTRODUCTION

1.1 Motivations

Depth sensors have increasingly become an integral part of our daily lives. With the advent of Microsoft Kinect [9], pmd CamCube 3.0 [10] and WAVI Xtion [11], the depth cameras are being used extensively in applications such as gaming, action recognition, scene understanding, and virtual reality. Depth cameras use a new technology that captures depth information of the 3D scene in real time by projecting infrared rays onto the scene. In recent years this technology has not only improved in quality but has also become cheaper and more compact. Some examples of these sensors are shown in Fig. 1.1. Due to their popularity, depth camera research has exploded in the last few years with discoveries of various applications in computer and robot vision [12–18], including object recognition. The first smart-phone with a depth camera was unveiled recently [19], paving the way for consumer friendly depth based applications in the near future.

![Examples of depth sensors](image)

(a) Microsoft Kinect (b) PMD camcube (c) ToF IR sensor

Figure 1.1: Examples of depth sensors.

This work is based on projective geometry. Projective geometry is similar to the pinhole camera model that deals with how an image is formed. This concept can be traced to Mo-Ti in China around 400BC and was first built by Arab physicist Ibn al-Haytham around 1000AD. In projective geometry, information such
as length (size) and angles between objects is not preserved. However, straight lines are still straight and parallel lines instead of being parallel may meet at 2D points called vanishing points. An example of scene formation using projective geometry is shown in Fig. 1.2.

![Image of projective geometry](image.png)

Figure 1.2: Image formation using projective geometry.

The depth images provided by these cameras are significantly different from the color images that standard color cameras provide. While the color cameras provide intensity information at each pixel, the depth cameras provide the distance information of the surroundings at each pixel. A color image is visually appealing and intuitive to a human being. In contrast, the depth image does not seem to be very helpful to human beings as it only provides us the depth information at each pixel. Human beings, in general, estimate the depth of the scene using visual cues in images and are particularly sensitive to color discontinuities which a depth image lacks. For example, in Fig. 1.3, if one looks at the depth image alone, one can perhaps identify a cone lying on top of the box. Most will guess the other objects to be a circular and rectangular plate which in reality are a football and a book respectively. However, once we look at the color image of the same scene, one can identify a book named “Multi View Geometry”, a cone and a football lying on top of a box that has been covered with a highly textured floor mat. Human beings can estimate the size of the football, books, etc., from their previous experiences of handling and observing these objects. The human brain is also extremely adept at resolving occlusion issues in a scene. However, the same is not true for machine vision. Current computer vision algorithms cannot cleverly predict the size of objects or resolve occlusion issues as a human brain can. Thus, a depth image becomes extremely useful for such algorithms. If we know how far the object is from the camera whose parameters have been computed beforehand, we can compute the size of the objects quickly. These camera parameters are computed by performing camera calibration.
Depth and color sensors complement each other very well. While a color sensor provides intensity information at each pixel, a depth sensor provides distance at each pixel. Combining these two provides distance and intensity at every pixel and, theoretically, we can construct accurate 3D models and scenes using this information. We need to perform two main steps to merge the depth and color information. First, we need to self-calibrate the depth sensor, and second, we need to perform cross-calibration with the color sensor. Depth sensors usually have a very low resolution and suffer from significant noise in depth measurements, particularly at edges. The standard calibration scheme is not sufficient to self-calibrate the depth sensor due to its limited resolution and presence of significant noise.

Once we have a good mapping between the color and depth sensor, we are able to create real-time realistic and interactive 3D models of the scene. The fusion of color and depth information has truly created a revolution in 3D imaging. The depth cameras can be mounted on robots, unmanned aerial vehicles (UAVs), drones, and quadcopters and capture the scene geometry and camera trajectory in real-time. This has created a need for automatic detection of interesting objects present in the scene. Such applications can not only assist in navigation of these devices but also help in localizing and identifying the objects that are present in the scene.

Depth cameras such as Kinect have certain limitations. They require external power source, are heavy and have a short range (7 m). Thus, in some cases using depth sensors (Fig.1.1(c)) is more efficient. These sensors can be connected directly to a circuit board, are extremely light and have a higher working range (14 m). These sensors provide a scalar, one pixel, distance information compared to a 2D array distance information provided by depth cameras. These sensors are mainly used on UAVs that cannot provide external power source or carry heavy loads. These sensors are mainly used to measure the distance of the UAV from the surroundings. If we know the scene geometry, this information can be extremely useful to estimate the 3D location of the UAV in the scene.

In summary, the main motivation in this thesis is to improve the depth sensor measurements and use these denoised measurements for various practical applications. First, a non-linear optimization based depth self-camera calibration scheme is presented. Next, the depth information along with the captured scene and camera trajectory is used to automatically output precise 3D object proposals in a given scene. Finally, we use depth sensors along with known scene geometry to
stitch images captured by a camera moving forward in a spirally rotating fashion. We also create an immersive 3D environment of the scene in Unity to allow users to navigate around in the scene and inspect the scene as per their convenience.

1.2 Problem Statement

This thesis investigates a new method for improving depth camera self-calibration and using the improved measurements for practical applications. In particular, the following problems are addressed:

1. Pre-processing the depth information to denoise it and the denoised depth-information to perform depth camera self-calibration.

2. Integrating the captured depth information with 2D color images to build a 3D global map of the environment and automatically estimate 3D bounding boxes for the objects of interest that may be present in a given scene by exploiting the multi-view scene geometry.

3. Exploiting scene geometry along with depth sensors by utilizing structure-from-motion (SfM) coupled with bundle adjustment (BA) to create cylindrical images of a given scene, such as a tunnel and an underpass, where the camera moves forward in a spirally rotating fashion.
1.3 Thesis Contributions and Summary

The remainder of this thesis is structured as follows.

Chapter 2 focuses on depth camera self-calibration. We exploit the noise properties of PMD devices to denoise depth measurements and perform camera calibration using the denoised depth as an additional set of measurements. Our synthetic and real experiments show that our depth denoising and depth based calibration scheme provides significantly better results than traditional calibration methods.

In Chapter 3, we leverage depth information per frame and multi-view scene information to obtain accurate 3D object proposals. Using standard metrics, such as Precision-Recall curve and detection rate, we show that our approach is significantly more accurate than the current state-of-the-art techniques.

In Chapter 4, we exploit structure-from-motion (SfM) along with bundle adjustment (BA) and known scene geometry to robustly stitch images captured in a cylindrical tunnel where the camera moves forward in a spiral fashion. We investigate the maximum allowable speed of UAV in the tunnel for a full panoramic stitch of the tunnel. We discuss in detail how the captured images of the inner surface of the tunnel are combined to create a composite panoramic image. This panoramic image is then textured onto a 3D cylinder in Unity for users to navigate through and inspect specific parts of the tunnel.

Finally in Chapter 5, we conclude our thesis and provide remarks about the results and potential future applications.
CHAPTER 2

CALIBRATION OF DEPTH CAMERAS

2.1 Introduction

An important recent development in visual information acquisition is the emerging low-cost and fast cameras for measuring depth [20]. With the advent of Microsoft Kinect [9], PMD CamCube 3.0 [10] and WAVI Xtion [11], the depth cameras are being used extensively in applications such as gaming and virtual reality. These cameras measure the time-of-flight (TOF) of infrared light at video frame rates. With the development of the TOF cameras, the structural information about the scene can be captured at high speed, and it can be incorporated in many applications due to their mobility. Obtaining such information is crucial in many 3D applications; examples include image based rendering, 3D reconstruction, motion capture, and scene perception.

Unfortunately, the imaging capabilities of current TOF cameras are very limited when compared to conventional color sensors. They can only provide low-resolution intensity images and depth maps with a high noise level that contain significant lens distortion. A brief overview of the capturing process of a TOF camera is provided in Appendix A.

Camera calibration refers to performing a set of controlled experiments to determine initial parameters of the camera that affect the imaging process of the scene. Without an accurate calibration, one will not be able to get a good 3D representation of the scene. Thus, camera calibration is an extremely important step in 2D and 3D computer vision. The traditional calibration scheme is not sufficient to self-calibrate the depth camera due to the limited resolution of depth cameras and presence of significant noise. Both the camera calibration and the depth denoising need to be significantly improved to obtain satisfactory calibration results.

In this work, we propose a novel algorithm that takes in few calibration images and uses them to simultaneously denoise and calibrate TOF depth cameras.
The main contributions of this work are:

1. Use depth planarization in 3D to denoise the depth at each corner pixel.

2. Integrate these improved depth measurements along with the corner pixel information to estimate the calibration parameters using a non-linear estimation algorithm.

We demonstrate that our framework estimates the intrinsic and extrinsic calibration parameters more accurately using fewer images and corners than are needed for traditional camera calibration. We evaluate our approach on both a synthetic dataset where groundtruth information is available, and real data taken from a PMD (photon mixing device) camera. In both cases we demonstrate that our proposed framework outperforms traditional calibration technique without significant increase in computational complexity. Moreover, our framework uses fewer images and corner points making it easier to use for the general public.

This chapter is organized as follows. First, we present the relevant research in Section 2.2. Standard calibration scheme and other related background material will be presented in Section 2.3. Our proposed method to denoise depth measurements will be introduced in Section 2.4. Section 2.4 will provide a detailed explanation of how to integrate the denoised depth measurements to improve depth camera calibration. Theoretical and empirical results will be provided in Section 2.5. Finally, the chapter will be concluded in Section 2.6.

## 2.2 Calibration Parameters

### 2.2.1 Color Camera Calibration

A lot of work has been done in the computer vision and photogrammetry community [21], [22], [23], [24] to perform color camera calibration.

Zhang [21, 22] suggests that intrinsic parameters of the color cameras can be calibrated using some known geometry in the scene. The relationship between measurement points in the images captured and their known 3D positions in space can be exploited to estimate the intrinsic and extrinsic calibration parameters. This scheme is utilized heavily in most common calibration packages such as MATLAB Camera calibration Toolbox [25] and OpenCV [4]. This traditional approach
uses a set of checkerboard images taken at various positions and exploits planar geometry to estimate the calibration parameters.

2.2.2 PMD Camera Calibration

Since PMD cameras are relatively new, most of the current approaches borrow heavily from traditional camera calibration technique. Kahlmann et al. [26] explore the depth related errors at various exposure times. They use a look-up table to correct for the depth noise. This approach is time consuming and entails creating a look-up table each time.

Linder and Kolb [27] use a controlled set of measurements to perform depth camera calibration. The checkerboard is put on a very precise optical measurement rack which is moved 10 cm away from the camera iteratively and this prior knowledge is used to correct the depth at corner points. Schiller et al. [28] use two CCD cameras along with a depth camera and use an analysis by synthesis approach to estimate the systematic depth measurement error and calibration parameters simultaneously. The intrinsic and extrinsic calibration parameters are computed using only a single image. However, their approach ignores any kind of lens distortion and depth noise present in the dataset, and hence is quite inaccurate.

Fuchs and Hirzinger [29] use a color and a depth camera rigidly set up on a robotic arm and move the arm with a pre-determined set of poses to estimate the calibration parameters using a checkerboard. They do not estimate the lens distortion parameters assuming the camera contains insignificant radial and tangential distortion. Beder and Koch [30] estimate the focal length and extrinsic parameters of the PMD camera using the intensity map and depth measurements from a single checkerboard image. They assume the camera to be distortion free with optical center lying at the image center.

2.2.3 Kinect Camera Calibration

Kim et al. [31] present a method to calibrate and enhance depth measurements for Kinect. They project the depth onto color sensor’s camera plane and use a weighted joint bilateral filter considering the color and depth information at the same time to reduce the depth noise. Herrera et al. [32] use a depth and color camera pair to perform camera calibration using a planar checkerboard by uti-
lizing the camera’s depth to improve the calibration. However, they assume the
depth camera to be distortion free and only estimate two disparity mapping related
parameters for the Kinect camera. Hence, their method is unable to estimate the
actual intrinsic parameters of the depth camera.

In a recent work, Herrera et al. [33] propose an algorithm that performs cali-
bration with Kinect depth sensor and two color cameras using 60 checkerboard
images. While their algorithm accounts for depth noise, they assume the depth
sensor to be distortion free. Our approach closely resembles theirs. However, we
use a PMD camera that contains significant photon noise and has a much lower
resolution than Kinect.

Most of these techniques either require multiple cameras or a controlled set-up
to exploit some prior knowledge to estimate the calibration parameters. More-
over, most of these approaches ignore lens distortion which is significant in PMD
cameras. We aim to provide a simple approach that estimates lens distortion and
performs calibration while simultaneously denoising the depth map by exploiting
scene planarity using as few images and corners as possible.

2.3 Color Camera Calibration

In this section, we describe the basics of traditional color camera calibration and
a commonly used algorithmic approach to estimate the camera calibration param-
eters.

2.3.1 Calibration Parameters

Camera calibration involves finding the two main sets of parameters - intrinsic
and extrinsic. Intrinsic parameters refer to the parameters of our camera such as
image center, focal length, and any lens distortion. As the camera moves in the
scene during the capturing process, its location and pose change with respect to
the world coordinate frame. Extrinsic parameters refer to the pose and location of
the camera per image frame.

**Notation Used:** We represent a 3D corner point in camera coordinate frame as $x_{w}$ and the corresponding 3D corner point in world coordinate frame as $x_{c}$. We capture $N$ images containing $M$ corners per image. The corners of each image grouped together in a matrix in camera and world coordinate frame are referred as
\(X_c\) and \(X_w\) respectively. \(X^j\) refers to all the corners in \(j^{th}\) image and \(X^{i,j}\) refers to the \(i^{th}\) corner in \(j^{th}\) image. \(x = [u, v]^T\) represents a corner pixel in the camera plane. Similarly, \(x^{i,j}\) represents the \(i^{th}\) corner pixel in the \(j^{th}\) image frame. The measured quantities such as corners are referred to as \(\tilde{x}\) while the estimated parameters using the optimization are referred as \(\hat{x}\).

**Intrinsic Camera Calibration Parameters:** Intrinsic camera calibration parameters consist of the image center, focal length of camera, scaling factors of row and column pixels, skew factor and any lens distortion in the camera. The intrinsic calibration matrix of a camera, \(K\), contains five parameters - focal length in \(x\) and \(y\) directions, \([f_x, f_y]^T\); skew \(s\); and the location of optical center, \([c_x, c_y]^T\), are defined as:

\[
K = \begin{bmatrix}
  f_x & s & c_x \\
  0 & f_y & c_y \\
  0 & 0 & 1
\end{bmatrix}
\]  

(2.1)

Here \(f_x = f \cdot m_x\) and \(f_y = f \cdot m_y\) represent the focal length of camera in pixels in horizontal and vertical direction respectively, where \(f\) is the focal length of camera and \(m_x\) and \(m_y\) are the number of pixels per unit distance in \(x\) and \(y\) direction respectively. The parameter \(s\) represents the skew parameter, which is generally zero for non fish-eye lenses. \(c_x\) and \(c_y\) are known as the principal point or the optical center of the camera. The optical center represents the image center in pixels and may not always lie at the center of the image. We represent a 3D point in camera coordinate frame as \(x_c\). The 3D points are projected onto camera plane at the normalized pixel position, \(x_n = [x_n, y_n]^T\) as:

\[
\begin{bmatrix}
x_n \\
y_n
\end{bmatrix} = \begin{bmatrix}
x_{c,1}/x_{c,3} \\
x_{c,2}/x_{c,3}
\end{bmatrix}
\]  

(2.2)

In theory, every lens can be perfectly parabolic, but in practice this is not the case. Usually a lens is more “spherical” which leads to radial distortion as shown in Fig. 2.1a. The radial distortion is sometimes known as “barrel” or “fish-eye” distortion. The radial distortion causes the rays farther from the optical center to bend more than the rays nearer to the optical center. This leads to the “barrel effect”. While this distortion is insignificant in high-end cameras, cheap cameras like web-cams often suffer from significant radial distortion. In general, the radial
Figure 2.1: The two types of distortions present in an imaging system are radial and tangential distortion. Radial distortion in (a) caused the rays farther from the optical center to bend more than the rays nearer to the optical center [4]. Tangential distortion in (b) happens due to manufacturing defects where the imaging plane of the camera is not perfectly parallel to the lens [4].

distortion is induced by using three parameters, \(k_1, k_2,\) and \(k_3\):

\[
\begin{align*}
    x_r &= x_n (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\
    y_r &= y_n (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)
\end{align*}
\]  

(2.3)

(2.4)

where \([x, y]^\top\) is the original pixel value of a point, \([x_r, y_r]^\top\) is its distorted pixel value, and \(r\) refers to the distance of the point from the optical center. The parameters \(k_1, k_2,\) and \(k_3\) are computed by ensuring that the co-linear points in the scene are indeed co-linear in the image as well.

Another common distortion seen in some cameras happens when the imager and lens do not align properly, which results in tangential distortion as shown in Fig. 2.1b. This usually happens due to manufacturing defects where the imaging plane of the camera is not perfectly parallel to the lens. Tangential distortion is introduced by using two additional parameters, \(k_4\) and \(k_5\), as follows:

\[
\begin{align*}
    x_t &= x_n + 2k_4 x_n y_n + k_5 (r^2 + 2x_n^2) \\
    y_t &= y_n + 2k_5 x_n y_n + k_4 (r^2 + 2y_n^2)
\end{align*}
\]  

(2.5)

(2.6)

where \([x, y]^\top\) is the original pixel value of a point, and \([x_t, y_t]^\top\) is its distorted pixel value. These five distortion parameters are usually bundled together as \(k_e = [k_1, k_2, k_3, k_4, k_5]^\top\). The distorted pixel value of this point, \(x_d\), is obtained
after adding the distortion model as:

\[
\begin{bmatrix}
  x_d \\
  y_d
\end{bmatrix} = \begin{bmatrix}
  x_n(1 + k_1r^2 + k_2r^4 + k_3r^6 + 2k_4y_n) + k_5(r^2 + 2x_n^2) \\
  y_n(1 + k_1r^2 + k_2r^4 + k_3r^6 + 2k_5x_n) + k_4(r^2 + 2y_n^2)
\end{bmatrix}
\]  

(2.7)

Here, \( r \) refers to the distance of the point from optical center. Eventually, the final pixel position, \( x_p \), recorded by the camera is obtained by using the intrinsic calibration matrix as:

\[
\begin{bmatrix}
  x_p \\
  y_p \\
  1
\end{bmatrix} = K \begin{bmatrix}
  x_d \\
  y_d \\
  1
\end{bmatrix}
\]  

(2.8)

**Extrinsic Camera Calibration Parameters:** There are various ways to perform color camera calibration with lens distortion taken into account. A widely used calibration toolbox [34] uses a planar checkerboard pattern with \( M \) corners to perform the calibration. The user holds a checkerboard in front of the camera and takes \( N \) images with the checkerboard held in various positions. An example of an image captured by a color and depth camera is shown in Fig. 2.2. The 3D points that lie on the checkerboard are expressed in terms of a world coordinate frame, \( x_w \). For every \( j^{th} \) image, the two coordinate frames camera (\( x_c \)) and world (\( x_w \)), are related via a rotation matrix, \( R \), and translation vector, \( t \), as:

\[
x_c = Rx_w + t
\]  

(2.9)

where

\[
R = \begin{bmatrix}
  r_{11} & r_{12} & r_{13} \\
  r_{21} & r_{22} & r_{23} \\
  r_{31} & r_{32} & r_{33}
\end{bmatrix} ; \quad t = \begin{bmatrix}
  t_x \\
  t_y \\
  t_z
\end{bmatrix}
\]  

(2.10)

Even though \( R \) contains nine elements, it only has three degrees of freedom - one each for rotation in \( xy \), \( yz \) and \( xz \) plane. The constraints on \( R \) come from the fact that a unitary normal matrix, i.e. \( R^T R = I_{3\times3} \). We define a rotation vector, \( r \), consisting of three parameters as:

\[
r = \begin{bmatrix}
  r_x \\
  r_y \\
  r_z
\end{bmatrix}
\]  

(2.11)
We use the Rodrigues rotation formula in Eq. 2.12 to obtain the rotation matrix $R$:

$$
\theta = ||r||_2 \\
u = r/\theta \\
U_\times = \begin{bmatrix}
0 & -u(3) & u(2) \\
u(3) & 0 & -u(1) \\
-u(2) & u(1) & 0
\end{bmatrix}
$$

$$
R = I_{3\times3} + \sin \theta \cdot U_\times + (1 - \cos \theta) \cdot U_\times^2
$$  \hspace{1cm} (2.12)

where $U_\times$ is called the skew symmetric matrix of the vector $u$.

The rotation matrix and translation vector, $\{R^i, t^i\}$, are bundled together for each image and calibrated together with the intrinsic parameters. We denote all the calibration parameters $(K, k_c, \{R^1, t^1, R^2, t^2, \ldots, R^N, t^N\})$ as $V$.

Figure 2.2: A sample color and depth image captured by PMD camcube for the calibration process. The checkerboard is moved around the scene and captured using a depth camera.

2.3.2 Problem Formulation

As we see in Fig. 2.2, the checkerboard consists of $M$ interior points. In general, the corners are estimated to a pixel precision using Harris corner detection \cite{4} that computes the Hessian matrix of the image. A sub-pixel corner estimation method
is used to improve the precision of the corners up to 0.1 pixels. The corners are
extracted for all \( N \) images and saved as \( x_i^j \) where \( i \) refers to the corner number
and \( j \) refers to the image number.

\[
x_i^j = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}^{i,j} \quad \forall i = \{1, 2, ..., M\}, \quad \forall j = \{1, 2, ..., N\}
\]

The exact size of each block in the checkerboard is known beforehand to be \( b \) m.
For each image, the world co-ordinate frame is assumed to start at the left top-most
interior corner of the checkerboard with \( x \) axis going to the right horizontally and
\( y \) axis going down vertically. The checkerboard is placed in the \( xy \) plane resulting
the \( z \) co-ordinate for all the corners to be 0. The 3D checkerboard corners, \( C \), for
each image are as follows:

\[
C_x = \begin{bmatrix}
0 & b & 2b & \ldots & (m-1)b \\
0 & b & 2b & \ldots & (m-1)b \\
\vdots & \vdots & \ddots & \vdots \\
0 & b & 2b & \ldots & (m-1)b
\end{bmatrix}, \quad C_z = \begin{bmatrix}
0 & 0 & \ldots & 0 \\
0 & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 0
\end{bmatrix}
\]

\[
C_y = \begin{bmatrix}
0 & 0 & \ldots & 0 \\
b & b & \ldots & b \\
\vdots & \vdots & \ddots & \vdots \\
(n-1)b & (n-1)b & \ldots & (n-1)b
\end{bmatrix}
\]

where \( C_x, C_y, \) and \( C_z \) are the \( x, y, z \) coordinates of the corner pixels. \( m, n \)
are the number of corners in horizontal and vertical directions respectively, i.e.
\( M = m \times n \). The 3D checkerboard corners are vectorized into \( 3 \times M \) matrix, \( X_w \),
with \( k^{th} \) column referring to the coordinates of \( k^{th} \) corner. Since we follow the
same procedure throughout all the checkerboard images, \( X_w \) remains unchanged
for the entire calibration step.

For every \( j^{th} \) image frame, we use a rotation matrix \( R^j \) and a translation vec-
tor \( t^j \) to transform these 3D co-ordinates in world frame of reference, \( X_w \), into
camera’s frame of reference, \( X_c \). The rotation matrix and translation matrix each
have three degrees of freedom and are unique for each image. Hence, overall we
Figure 2.3: The checkerboard is transformed per frame from world coordinate frame to camera coordinate frame using the extrinsic camera parameters. Then it is projected onto camera plane using intrinsic camera parameters.

have $N$ such rotation matrices and translation vectors.

$$X_c^j = R^j X_w^j + t^j \quad \forall j = \{1, 2, ..., N\}$$

(2.14)

Each 3D point in camera coordinate frame, i.e. $X_{ij}$, is referred to as $x_c$. These 3D points are mapped onto the camera’s image plane using the intrinsic parameters $K$ and $k_c$ using Eqs. 2.2-2.8 as shown in Fig. 2.3.

2.3.3 Initialization of Calibration Parameters

**Intrinsic Calibration Parameters**: The principal point $[c_x, c_y]$ is initialized at the center of the image. The distortion parameters, $k_c$, are assumed to be zero. The focal length in $x$ and $y$ directions is assumed be identical for the initialization process. The focal length is initialized using vanishing points. For each image, two vanishing points $v_1, v_2$ are computed using two orthogonal directions. Let $x_{w1}$ and $x_{w2}$ be two points lying on $X$ and $Y$ directions in 3D plane. Their projection
onto camera plane is obtained by:

\[ v^i = \lambda^i K \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} x^i_w \\ 0 \end{bmatrix}, \quad \forall i = \{1, 2\} \]

\[ = \lambda^i K R x^i_w \]

Since \( K \) and \( R \) are invertible matrices, we can obtain \( x^i_w \) as:

\[ x^i_w = \frac{1}{\lambda^i} R^{-1} K^{-1} v^i, \quad \forall i = \{1, 2\} \quad (2.15) \]

Using the orthogonality principle \((x^1_w)^\top \cdot x^2_w = 0\), we get:

\[
\left(\frac{1}{\lambda^1} R^{-1} K^{-1} v^1\right)^\top \cdot \frac{1}{\lambda^2} R^{-1} K^{-1} v^2 = 0,
\]

\[
\frac{1}{(\lambda^1 \cdot \lambda^2)} (v^1)^\top K^{-1} R^{-1} R K^{-1} v^2 = 0,
\]

\[
(v^1)^\top K^{-1} K^{-1} v^2 = 0 \quad (2.16)
\]

We stack the measurements for \( N \) images and obtain the best estimate for focal length using least mean square (LMS) approach.

**Extrinsic Calibration Parameters:** The rotation and translation parameters are estimated per frame. Thus, we will only deal with one image here to avoid complicated indices in the equations. We initialize extrinsic parameters in a two-step process.

First, initial deterministic values of the extrinsic parameters \( \hat{r} \) and \( \hat{t} \) are estimated using homography between the corners of checkerboard in world coordinate frame \( C \), and camera plane \( x_p \). Once the extrinsic parameters are initialized, they are further refined using a local optimization in the second step.

Since the \( z \) coordinate of the corner points in world coordinate frame is zero and the distortion parameters are initialized to be zero, we can simplify the transformation and projection from 3D world coordinate frame to 2D camera plane as:
\[ \begin{align*}
  x^p_i &= \lambda^i K \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} C_x \\ C_y \\ 0 \\ 1 \end{bmatrix} = \lambda^i K \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \\
  x^i_p &= \lambda^i H \begin{bmatrix} C_x \\ C_y \\ 1 \end{bmatrix}
\end{align*} \] (2.17)

where \( r_i \) are the three columns of the rotation matrix \( R \). The \( 3 \times 3 \) matrix \( H \) is known as the homography matrix. Using \( M \) corners, we obtain \( 3M \) measurements for \( (9 + M) \) variables. Thus, we can solve for \( H \) as long as we have a sufficient number of corner points per image. Once we estimate the homography matrix using LMS, we can compute the extrinsic parameters:

\[
\begin{align*}
  r_1 &= K^{-1} h_1 \\
  r_2 &= K^{-1} h_2 \\
  r_3 &= r_1 \times r_2 \\
  t &= K^{-1} h_3
\end{align*}
\]

where \( h_i \) are the three columns of the homography matrix \( H \). This provides us with a good initialization of the extrinsic camera calibration parameters. We can express Eq. 2.17 for a corner point, \( x^i_w = \begin{bmatrix} u^i \\ v^i \\ 0 \end{bmatrix} \), as:

\[ \begin{bmatrix} x^i_p \\ 1 \end{bmatrix} = \lambda^i K x^i_c = \lambda^i K [R x^i_w + t] \quad \forall i = \{1, 2, ..., M\} \]

\[
\begin{align*}
  x^i_p &= \begin{bmatrix} f_x x^i_{c,1} + c_x \\ f_y x^i_{c,2} + c_y \end{bmatrix} = \begin{bmatrix} f_x x^i_{c,1} + c_x \\ f_y x^i_{c,2} + c_y \end{bmatrix} \\
  &= \begin{bmatrix} f_x (t_x + r_{11} u^i + r_{12} v^i) \\ f_y (t_y + r_{21} u^i + r_{22} v^i) \end{bmatrix} + c_x \\
  &= \begin{bmatrix} f_x (t_x + r_{11} u^i + r_{12} v^i) \\ f_y (t_y + r_{21} u^i + r_{22} v^i) \end{bmatrix} + c_y
\end{align*}
\]

17
where

\[
\begin{bmatrix}
  x^i_{c,1} \\
  x^i_{c,2} \\
  x^i_{c,3}
\end{bmatrix} = \begin{bmatrix}
  t_x + r_{11}x^i + r_{12}y^i \\
  t_y + r_{21}x^i + r_{22}y^i \\
  t_z + r_{31}x^i + r_{32}y^i
\end{bmatrix}
\]

In the second step, we use these initial estimates in an iterative method that minimizes the projection error between projected corners \(x_p\), and the corners computed in the image \(\tilde{x}_p\). Using the corner information can transform and project the 3D corners onto camera plane using Eq. 2.17. Since we know the pixel measurements for each corner, we can estimate the extrinsic calibration parameters by minimizing over the following objective function:

\[g^i = (x_p^i - \tilde{x}_p^i)\] (2.18)

\[\hat{R}, \hat{t} = \arg\min_{R, t} \sum_{i=1}^{M} ||g^i||_2^2\] (2.19)

where \(x_p^i\) and \(\tilde{x}_p^i\) refer to the \(i^{th}\) projected and measured corner pixel in camera plane respectively. Our goal is to estimate the values of \(\hat{R}\) and \(\hat{t}\) such that they minimize the 2D projection error. This is a highly non-linear optimization problem. The Gauss-Newton method [35] is used commonly to minimize this error, which effectively replaces the non-linear least squares problem by a sequence of linear least squares problems whose solutions converge to the solutions of the original non-linear problem. Like all methods based on Newton’s method, the Gauss-Newton method may fail to converge unless it is started close enough to the solution. So as long as our initial estimates for \(\hat{R}\) and \(\hat{t}\) are close to the actual solution, the method is guaranteed to converge. However, the method will sometimes fail to converge due to noisy estimation of corners or bad initialization. In such cases, the corresponding images and their corner information are removed from the calibration process.

To use the Gauss-Newton method, we first need to compute the Jacobian matrix w.r.t. the rotation and translation parameters denoted by \(q\):

\[q = [r_x, r_y, r_z, t_x, t_y, t_z]^T\] (2.20)

The gradient per corner, denoted by \(\frac{\partial (x_p)}{\partial q}\), is a \(2 \times 6\) matrix. This matrix is commonly known as the Jacobian matrix \(J(x_p)\). The Jacobian per corner point is
shown below:

\[
J_t(x_p) = \begin{bmatrix}
\frac{f_x}{x_c,3} & 0 & -\frac{f_x}{x_c,3} \\
0 & \frac{f_y}{x_c,3} & -\frac{f_y}{x_c,3} \\
\end{bmatrix} \quad J_R(x_p) = \begin{bmatrix}
\frac{f_z}{x_c,3} & 0 & -\frac{f_z}{x_c,3} \\
0 & \frac{f_y}{x_c,3} & -\frac{f_y}{x_c,3} \\
\end{bmatrix}
\]

where \( J_t(x_p) \) and \( J_R(x_p) \) are the Jacobian matrices for the translation and rotation parameters respectively. Using chain rule on Rodriguez formula gives us:

\[
J_r(x_p) = J_R(x_p) \frac{\partial R}{\partial r}
\]

where \( \frac{\partial R}{\partial r} \) is:

\[
\frac{\partial R}{\partial r} = \begin{bmatrix}
\alpha r_x (r_x^2 + r_z^2) & \alpha r_y (r_y^2 + r_z^2) - 2\delta r_y & \alpha r_z (r_z^2 + r_y^2) - 2\delta r_z \\
-\alpha r_x^2 r_y - \beta r_x r_z + \delta r_y & -\alpha r_x^2 r_y - \beta r_x r_z + \delta r_z & -\alpha r_x^2 r_y - \beta r_x r_z + \gamma \\
-\alpha r_x^2 r_z + \beta r_x r_y + \delta r_z & -\alpha r_x^2 r_z + \beta r_x r_y + \delta r_z & -\alpha r_x^2 r_z + \beta r_x r_y + \delta r_z \\
\alpha (r_x^2 r_z + r_y r_x^2) - 2\delta r_x & \alpha (r_x^2 r_z + r_y r_x^2) & \alpha (r_x^2 r_z + r_y r_x^2) - 2\delta r_x \\
\alpha (r_x r_y r_z + \beta r_x^2 + \gamma) & \alpha (r_x r_y r_z + \beta r_x^2 + \gamma) & \alpha (r_x r_y r_z + \beta r_x^2 + \gamma) \\
-\alpha r_x r_y r_z + \beta r_x^2 r_y + \delta r_z & -\alpha r_x r_y r_z + \beta r_x^2 r_y + \delta r_z & -\alpha r_x r_y r_z + \beta r_x^2 r_y + \delta r_z \\
\alpha (r_x^2 + r_y r_x^2) - 2\delta r_x & \alpha (r_x^2 + r_y r_x^2) - 2\delta r_x & \alpha (r_x^2 + r_y r_x^2) - 2\delta r_x \\
\end{bmatrix}
\]

where

\[
\alpha = \frac{(2 - 2\cos \theta - \theta \cdot \sin \theta)}{\theta^4} \\
\beta = \frac{(\sin \theta - \theta \cdot \cos \theta)}{\theta^3} \\
\gamma = \frac{\sin \theta}{\theta} \\
\delta = \frac{1 - \cos \theta}{\theta^2}
\]

The overall Jacobian \( J(x_p) \) matrix per corner is expressed as:

\[
J(x_p) = [J_r(x_p), J_t(x_p)]
\]
**Algorithm 1** Local optimization process.

1: \( r_0 \leftarrow r_{\text{init}} \)
2: \( t_0 \leftarrow t_{\text{init}} \)
3: \( q_0 \leftarrow [r_0; t_0] \)
4: for \( k = 1 : \text{max\_iterations} \) do
5: \((J^\top J)s_k = -J^\top g\)
6: \( q_k = q_{k-1} + s_{k-1} \)
7: if \( |q_k - q_{k-1}| \leq \epsilon \) then
8: terminate
9: end if
10: end for

Using \( M \) such corners gives us:

\[
J = \begin{bmatrix}
J(x_1^p) \\
J(x_2^p) \\
\vdots \\
J(x_M^p)
\end{bmatrix}
\quad (2.21)
\]

A non-linear optimization algorithm such as gradient-descent or Levenberg Marquardt Algorithm (LMA) is used to estimate the rotation and translation parameters per frame. The main step for LMA based estimation is shown in Algorithm 1.

2.3.4 Final Estimation of Calibration Parameters

After obtaining good initial estimates for \([K, \{r^j, t^j\}], \forall \ j = \{1, 2, ..., N\}\), we perform a global optimization over all the images. Conceptually, this is similar to the local optimization step explained in the previous section. The main difference is that we now perform optimization over the intrinsic parameters as well and include all the corners for every image instead of performing the optimization independently per image.

\[
v = [K, k_e, \{r^j, t^j\}]^\top \quad \forall j = \{1, 2, ..., N\}
\quad (2.22)
\]

\[
\hat{v} = \underset{v}{\text{argmin}} \sum_{j=1}^{N} \sum_{i=1}^{M} (||g^{i,j}||_2^2)
\quad (2.23)
\]
Algorithm 2 The global optimization process

1: \( K_0 \leftarrow K_{\text{init}} \)
2: \( k_{c,0} \leftarrow k_{c,\text{init}} \)
3: \( r^{j,0} \leftarrow r^{j,\text{init}} \)
4: \( t^{j,0} \leftarrow t^{j,\text{init}} \)
5: \( v_0 \leftarrow [K_0, k_{c,0}, \{r_1^0, t_1^0\}, \ldots, \{r_N^0, t_N^0\}] \)
6: for \( k = 1 : \text{max iterations} \) do
7: \( (J^\top J)s_k = -J^\top g \)
8: \( v_k = v_{k-1} + s_{k-1} \)
9: if \( |v_k - v_{k-1}| \leq \epsilon \) then
10: terminate
11: end if
12: end for

The total number of parameters to be estimated now is \( 10 + 6N \). Once again a non-linear optimization framework such as LMA is used to estimate all the calibration parameters together as shown in Eq. 2.23. Algorithm 2 describes the iterative non-linear estimation process.

The general framework of standard camera calibration is demonstrated in Fig. 2.4.

![Figure 2.4: A modular approach for standard camera calibration.](image.jpg)
2.4 Depth Camera Calibration

Time-of-flight (ToF) cameras such as PMD camcube not only provide us an estimated intensity image but also another measurable quantity - depth at each pixel. This is the 3D scalar distance between the camera center and the point in 3D corresponding to that pixel. Depth at a pixel $x_n$ corresponding to the 3D point in camera coordinate frame $x_c$ is computed as:

$$d = ||x_c||_2 = \sqrt{||x_w||_2^2 + ||t||_2^2 + 2t^\top R x_w}$$

(2.24)

We use this additional set of measurements per corner pixel to perform the global optimization process by minimizing the following function using LMA with a user defined Jacobian matrix.

$$\hat{V} = \arg\min_V \sum_{j=1}^N \sum_{i=1}^M \left( \frac{||x^{i,j}_p - \tilde{x}^{i,j}_p||^2_2}{(\sigma_x^j)^2} + \frac{(d^{i,j}_p - \tilde{d}^{i,j}_p)^2}{(\sigma_d^j)^2} \right)$$

(2.25)

Here, $d^{i,j}_p$ refers to the estimated depth of the $i^{th}$ corner of the $j^{th}$ image and $\tilde{d}^{i,j}_p$ refers to the depth measured by the depth camera. We normalize the error terms in Eq. 2.25 with their respective variances, $\{(\sigma_x^j)^2, (\sigma_d^j)^2\}$, for every image as they have different measurement units.

**Depth Noise:** Like every sensing device, PMD also exhibits various error sources which affect the accuracy of depth information captured by it. There are three major sources of error in PMD cameras. First, the wiggling error is caused due to the hardware and manufacturing limitations. The outgoing signal is assumed to be perfectly sinusoidal. However, in reality, this signal is more “box-shaped” than sinusoidal [36]. Second, the flying-pixel error occurs at depth discontinuities. The depth at each pixel is computed by using four readings at each pixel. The information captured at each smart-pixel in PMD can come from either the background or foreground object, which leads to an unreliable depth measurement at these pixels. Third, the Poisson shot-noise error occurs due to reflectivity of the scene [36]. This inherent noise present in the capturing process leads to an unsteady 3D point cloud. The noise can be partly reduced by spatial averaging using bilateral filters, but we cannot use this process for applications requiring an accurate depth map as smoothing a depth map is highly undesirable. Thus, before we use the depth measurements, we pre-process the depth image to ensure that the depth at corner pixels is as accurate as possible.
Algorithm 3 Depth based calibration

1: procedure DEPTHBASEDCALLIB($x_w$, $x_p$, $d$, cSize)
2: $V \leftarrow \text{colorCalib}(x_w, x_p)$
3: $\hat{d} \leftarrow \text{planarizeDepth}(x_p, d, K, k_c)$
4: count $\leftarrow 0$
5: while (count $\leq$ maxIter & $\epsilon \geq$ threshold) do
6: $\hat{K} \leftarrow \text{updateK}(x_p, \hat{d}, K, k_c)$
7: $\hat{d} \leftarrow \text{planarizeDepth}(x_p, \hat{d}, \hat{K}, k_c)$
8: $\epsilon \leftarrow \text{errorIn3D}(x_p, \hat{d}, \hat{K}, k_c, cSize)$
9: count $\leftarrow$ count + 1
10: end while
11: $\hat{R'}, \hat{t'} \leftarrow \text{localOptim}(x_w, x_p, \hat{d}, \hat{K}, k_c, R', t')$
12: $\hat{k_c} \leftarrow \text{updateDistortion}(x_w, x_p, \hat{d}, \hat{K}, k_c)$
13: $\hat{V} \leftarrow \text{globalOptim}(x_w, x_p, \hat{d}, \hat{V})$
14: return $\hat{V}$
15: end procedure

2.4.1 Optimization Algorithm

In this section, we describe, step by step, how our calibration scheme works. Algorithm 3 delineates our depth based calibration process.

**Color Image Calibration (line 2):** We perform traditional calibration as described in Section 2.3. This provides us an initial estimate for the calibration parameters.

**Planarizing the Depth Image (line 3):** Since we only look at the interior corner points of a planar checkerboard, there is insignificant flying-pixel noise. Instead of denoising the depth measurement through spatial filtering, we employ prior knowledge about the scene, which is a checkerboard in our case. We account for wiggling error and reflectivity based noise by performing image segmentation and 3D plane estimation as shown in Fig. 2.5. We use the corner pixel information to segment out the white squares, where depth is more accurate than in the black squares. This is because the Poisson shot noise is higher in darker regions (black squares) compared to lighter regions (white squares) as seen in Fig. 2.6(a). We segment out the white squares and use their corresponding depth along with initial calibration parameter estimates to project the points in 3D. Thereafter, we use RANSAC along with gradient threshold to find the best plane using SVD. We estimate the depth at sub-pixel corners by finding the intersection of this estimated plane and a line passing through the sub-pixel corners when projected in 3D using
Figure 2.5: We use the interior corner along with intensity information to extract the white checkerboard regions. These regions do not suffer from Poisson noise and thus can be used to estimate the 3D checkerboard plane. This provides us a more accurate depth at the sub-pixel corners as seen in Fig. 2.6(b). The wiggling error is non-systematic and can lead to both under- and overestimation of depth [36]. We claim that the 3D planarization eliminates the wiggling error in these regions once we have enough white checkerboard regions. We denote this denoised depth as $\hat{d}$.

Figure 2.6: Checkerboards projected in 3D using (a) Original depth information (b) 3D planarization. The 3D planarization helps in removing the Poisson noise present in darker regions of the checkerboard making the calibration process more accurate.

**Updating $K$ (lines 6-8):** The calibration parameters provided by traditional calibration when using a small set of images and corners are very unreliable. Since
the calibration procedure involves using non-linear estimation, a good initialization of the calibration parameters is extremely important. Hence, it is critical to re-initialize these parameters before using them for global optimization. Due to the coupling of $K$ with $R^j$ and $t^j$, as seen in Eqs. 2.2-2.9, traditional calibration often fails to provide a good estimate for intrinsic calibration matrix as we lose a degree of freedom by projecting 3D coordinates onto the 2D camera plane. First, we use the estimated distortion parameters to obtain the normalized pixel positions for each corner, $x_n$. We use the denoised depth, $\hat{d}$, to obtain the 3D coordinates for each corner by projecting 2D corner locations in 3D:

$$x_c = \frac{\hat{d}}{\|K^{-1}x_n\|_2}K^{-1}x_n$$ (2.26)

Then, we use a non-linear optimizer to re-estimate $K$ by enforcing the projected checkerboard squares in 3D using the denoised depth data to be the same size as the actual checkerboard squares for each image:

$$\hat{K} = \arg\min_K \sum_{i=1}^{M} \sum_{l \in N(i)} (\|x_c^i - x_c^l\|_2 - cSize)^2$$ (2.27)

where $cSize$ refers to the checkerboard square size and $N(i)$ represents the neighbours of the $i^{th}$ corner. We repeat this process until this error is below a certain threshold. This provides us a reliable initial estimate for $K$ which is crucial for the optimization process.

**Re-initialization (lines 11-12):** We use the updated $K$ to re-initialize our extrinsic parameters in the same fashion as is done for the traditional calibration process. We also update the distortion parameters by assuming the remaining parameters as groundtruth and minimizing the objective function in Eq. 2.25.

**Global Optimization (line 13):** Finally, we bundle everything together and perform a global optimization using Eq. 2.25, using LMA as our non-linear solver with our new Jacobian matrix.

### 2.5 Experimental Results

In this section, we perform synthetic and real experiments on a PMD camera and compare our calibration scheme with the traditional calibration scheme.
Figure 2.7: Relative error in focal length for noisy synthetic data. We magnified the scale of the vertical axis in the cases of 9-36 corners to highlight the accuracy of our calibration scheme.

**Synthetic Data Results:** We synthesized a $12 \times 12$ checkerboard with 50 mm checker size containing 121 interior corners. We used up to 7 images and 36 corners for calibration. We added white Gaussian noise to corner pixels and depth data with a standard deviation of 0.01 pixels and 10 mm respectively to generate noisy data. This amount of noise resembles the noise present in real data in corner estimation and depth measurements captured by PMD cameras. We used varying subsets of 7 images and 36 corners to estimate the calibration parameters to highlight the fact that our approach outperforms the traditional approach when little information is available for calibration. We tested the calibration results on the entire checkerboard region (121 corners). Both the traditional and our calibration approaches achieved perfect results for the noiseless dataset when more than 9 corners and 3 images are available. Table 2.1 shows the mean 3D error as shown in Eq. 2.28 between the groundtruth corners and corners computed using the estimated calibration parameters from the two methods and groundtruth depth. Our approach outperforms traditional calibration in every test. Fig. 2.7 shows relative error in focal length ($=|\Delta f|/f_o$) for noisy synthetic data. Our approach consistently provided significantly better results than the traditional calibration approach. We observed similar improvements in optical center and extrinsic calibration parameters.

$$
\epsilon = \frac{1}{MN} \sum_{j=1}^{N} \sum_{i=1}^{M} \| \hat{x}_{w}^{i,j} - x_{w}^{i,j} \|_2
$$  

**Real Data (PMD) Results:** We used a checkerboard with 50mm checker size to capture 12 images using a PMD camera. Each checkerboard contains 70 corners. We used up to 7 images and 36 corners to estimate the intrinsic and extrinsic calibration parameters. We compare the focal length, $f$, obtained from both approaches to the manufactured focal length of the PMD camera, $[284.4, 284.4]^T$. 
pixels. We assume this value as groundtruth. As seen in Fig. 2.8, our approach consistently provides a reasonably accurate focal length while traditional calibration estimates a highly inaccurate focal length in most cases. One significant deviation from this behavior happens when only four corners are available for calibration. This is because the estimation process diverges as the initial estimates are far away from the ground truth where the non-linear estimation process (LMA) is known to fail frequently. However, once we use nine or more corners per image, our approach consistently provides significantly better results.

2.6 Conclusion

We presented a simple and accurate method to simultaneously denoise depth data and calibrate depth cameras. The presented method excels in estimating calibration parameters when only a handful of corners and calibration images are available, where the traditional approach really struggles. While this approach is simple and easily applicable, it still relies on using a checkerboard pattern to perform calibration. This approach has great potential to be extended to generic scenes where a user can exploit planarity present in the scene to perform calibration at home.
Table 2.1: Avg. 3D error between groundtruth corners and projected corners in mm.

<table>
<thead>
<tr>
<th># corners</th>
<th># images</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>traditional</td>
<td>53.6795</td>
<td>12.5422</td>
<td>8.6423</td>
<td>3.1773</td>
<td>1.8796</td>
</tr>
<tr>
<td></td>
<td>ours</td>
<td>7.9059</td>
<td>1.7206</td>
<td>0.8978</td>
<td>0.6689</td>
<td>0.4144</td>
</tr>
<tr>
<td>9</td>
<td>traditional</td>
<td>51.8946</td>
<td>10.7673</td>
<td>2.7336</td>
<td>3.3252</td>
<td>2.1771</td>
</tr>
<tr>
<td></td>
<td>ours</td>
<td>8.0193</td>
<td>1.7345</td>
<td>0.8591</td>
<td>0.4818</td>
<td>0.4220</td>
</tr>
<tr>
<td>16</td>
<td>traditional</td>
<td>151.9972</td>
<td>8.2549</td>
<td>6.6561</td>
<td>3.2660</td>
<td>1.7817</td>
</tr>
<tr>
<td></td>
<td>ours</td>
<td>37.3031</td>
<td>1.7453</td>
<td>0.8137</td>
<td>0.5875</td>
<td>0.4384</td>
</tr>
<tr>
<td>25</td>
<td>traditional</td>
<td>57.8784</td>
<td>8.0706</td>
<td>5.2105</td>
<td>2.5943</td>
<td>0.6113</td>
</tr>
<tr>
<td></td>
<td>ours</td>
<td>4.0092</td>
<td>1.5235</td>
<td>0.8008</td>
<td>0.5373</td>
<td>0.3934</td>
</tr>
<tr>
<td>36</td>
<td>traditional</td>
<td>40.4462</td>
<td>8.1659</td>
<td>2.9612</td>
<td>1.3889</td>
<td>0.8504</td>
</tr>
<tr>
<td></td>
<td>ours</td>
<td>4.2415</td>
<td>1.7081</td>
<td>0.8109</td>
<td>0.5640</td>
<td>0.4449</td>
</tr>
</tbody>
</table>


CHAPTER 3

LOCATING 3D OBJECT PROPOSALS: A DEPTH-BASED ONLINE APPROACH

3.1 Introduction

The rapid development of low-power unmanned aerial vehicles (UAVs), quadcopters, and drones has introduced a need for automatic detection of interesting objects present in a scene. Such applications can not only assist in navigation of these devices but also help in localizing and identifying the objects that are present in a scene. This chapter presents a framework where we integrate depth information captured using a depth sensing device with 2D color images, build a 3D global map of the environment and automatically estimate 3D bounding boxes for the objects of interest that may be present in a given scene by exploiting the multi-view scene geometry.

Depth cameras are increasingly becoming an integral part of our daily lives. Depth cameras are being used extensively in applications such as gaming and virtual reality. Due to their popularity, depth camera research has exploded in the last few years with discoveries of various applications in computer and robot vision [12–18], including object recognition. The first smart-phone with a depth camera was unveiled recently [19] paving the way for consumer friendly depth-based applications in the near future. While depth cameras have improved considerably since their inception, they still suffer from noise at depth discontinuities and in presence of specular objects [37], [36].

Object detection and recognition is an extremely important aspect of robot vision. In general, object detection is performed on image patches, traditionally sampled through a sliding window procedure [38]. Patch sampling has recently been made efficient by finding the top 1,000-5,000 window patches that have a high probability of containing an object. These patches are called object proposals [39–44]. Object proposals aim at finding a simple estimator that can efficiently pick up regions with object presence at a high recall rate. Object proposal tech-
niques are unique in that they are computationally fast and run on color images independently. This has helped in achieving real-time object recognition [45] albeit with the use of heavy parallel processing and expensive GPUs.

Song and Chandraker [46] present an impressive framework where they combine cues from structure for motion (SFM), object detection and plane estimation to estimate rigid 3D bounding boxes for moving cars in an outdoor real environment. Their approach assumes prior knowledge of number of cars present in a given scene and does not correct for the car dimensions in case they were detected incorrectly. They use a monocular slam to estimate camera pose and track 3D objects in the scene using feature points. Unfortunately, the object proposal methods specific for RGB images are sub-optimal for depth camera video input as no depth information is used and no dense temporal consistency is imposed. A robot can use the scene geometry to its advantage and maneuver around the scene to gain critical information about the objects that may be present in the scene [47]. Moreover, most of the 2D object proposal methods focus on high recall and output several object proposals around an object of interest. Selecting the best object boundary among these object proposal candidates is a difficult problem to solve.

In this work, we propose a real-time 3D object proposal technique for RGB-D video input with the main focus on precision.

We make the following contributions:

- We integrate depth information with state-of-the-art 2D object proposal techniques to improve object proposals per frame.

- During the scene capturing process, we exploit the indoor scene geometry to automatically remove any supporting planes from the regions of interest.

- We use the camera pose estimated by a depth based SLAM technique to efficiently register the frames to the global point cloud and output multi-view consistent 3D object candidates as the camera moves around in the scene.

To showcase our results, we perform density based 3D clustering on our top ranked points and display our proposed 3D objects bounded by tight 3D bounding boxes. One key aspect of our approach is that it is near real-time only using a single thread CPU; hence, our system is capable of being integrated on top of existing depth based SLAM methods.
This chapter is structured as follows: In Section 3.2, we review other works that are related to our research topic. In Section 3.3, we provide a brief overview of our problem formulation. Sections 3.4 and 3.5 present the proposed 2D depth based filtering and 3D fusion and refinement steps in detail, highlighting our contributions and observations at each stage. We report our results and comparison with other state-of-the-art methods in Section 3.6. Finally, in Section 3.7 we conclude this chapter and discuss future research directions.

3.2 Related Work

**Object Proposals:** A lot of work has been done recently on 2D object proposals. Traditionally, sliding windows are used along with a detector to identify objects in a given scene. Many of the state-of-the-art techniques in object detection have started using a generic, object-class agnostic proposal method that finds anywhere between 100 and 10,000 bounding boxes in an image. These areas are considered to have the maximum likelihood of containing an object. These methods vary widely in their approach. For example, BING [39] uses linear classifiers, CPMC [40] uses graph cuts, [48] uses graph cuts with an affinity function, MCG [41] uses normalized cuts, edge-boxes [42] uses random forests, [43] uses geodesic transform, and [44] uses super pixels in estimating the regions of interest.

Hosang et al. [49] provide a systematic overview of these object proposal techniques and tested these state-of-the-art algorithms such as edge-boxes [42], BING [39], geodesic [43], and MCG [41]. They demonstrated that most of these algorithms have limited repeatability. Even changing one pixel exhibited markedly different outcomes. They found edge-boxes and geodesic to give the best results for fast and high-quality object proposals. Based on these observations, we decided to use edge-boxes [42] as it is reported to be significantly faster and slightly more accurate than Geodesic object proposals.

**Simultaneous Localization and Mapping (SLAM) with Scene Geometry:** Choudhary et al. [50] proposed an online object discovery approach that extends standard SLAM to utilize the discovered objects for pose estimation and loop closures. They use an RGB-D camera mounted on a robot to create a 3D point cloud and use 3D feature descriptors, CSHOT [51], to find camera pose for every frame. They use 3D voxels to roughly segment out objects and use them as landmarks for loop closure.
Song and Chandraker [46] argued that the structure from motion (SFM) cues (3D points and ground plane) and object cues (bounding boxes and detection scores) complement each other. The former is accurate for nearby environments while the latter is more accurate for objects that are far away. They combine these two approaches and propose a framework for 3D object localization. Their system can be treated as an extension of Bundle Adjustment (BA) with object cues.

Pillai and Leonard [3] developed a monocular SLAM-aware system that used temporal information to output multi-view consistent object proposals using efficient feature encoding and classification techniques. They only utilize color information of the scene. Their approach works well if the objects are far apart so that the monocular SLAM method can provide a reliable semi-dense depth estimate for each view. However, their approach has the potential to fail on more crowded environments (lots of objects) or when objects are cluttered together in a small region. Using depth information enables us to handle these situations as we can exploit the 3D scene geometry to identify regions of interest, redundant vertical and horizontal planes and cluster them separately. We perform our evaluation on the same dataset for a direct comparison. We demonstrate that our results show significant improvement when compared with theirs. Another, and perhaps the most significant, difference in our approach is that we use depth images along with color images to output 3D object proposals while they compute 2D object proposals only.

**Single-view RGB-D Object Proposals:** A hypothesis worth testing is whether using only the depth images (single channel - D) or the depth images along with the standard RGB color channels (four channels - RGBD) can improve object proposals. References [52], [53], and [54] found depth based proposals to be inferior to color based proposals. They conclude that depth and color combined together can potentially give worse results when compared to using color alone due to the inherent noise present in depth images. We also tested this hypothesis on the UW-RGBD Scene Dataset [2]. Due to significant holes and noise present in the raw depth images, the results were extremely unsatisfactory.

Recently, a number of hybrid 2.5D approaches have been proposed, which transform depth input into more useful information such as “height above ground”, “angle with gravity” and “surface normals” [54] to be used together with color information per pixel for object proposals. Fouhey et al. [52] proposed an adaptive approach where they fuse depth with color using a maximum likelihood estimation technique. They fill the holes in the depth image using an iterative nearest
neighbor method and assign confidence based on whether the depth was originally measured or estimated in their method. They demonstrate that using the weighted scheme gave them better results than using just color or using color and depth without adaptive weights. We utilize the depth information to estimate the direction of gravity and horizontally planar points. This enables us to transform our 3D bounding boxes to align vertically with the global point cloud.

### 3.3 Proposed Approach

We start by giving an overview of the proposed algorithm. At high level, the proposed algorithm is designed to fuse the depth information with the generic object proposals obtained from using color images in an indirect manner. This enables us to exploit using 3D geometry of the scene without impacting the results due to noisy or unavailable depth information at various image pixels. Fig. 3.1 illustrates the basic setup of the problem studied in this chapter and an example result obtained by our algorithm. Our framework consists of a few novel modules in its pipeline (as shown in Fig. 3.2), and they will be presented in detail next.
Figure 3.2: Our proposed framework for 3D object proposals.

First, we introduce the initialization process in Sec. 3.3.1. Section 3.4.1 presents the representation for the local 2D heatmap. Sections 3.4.2 and 3.4.3 describe our depth based filtering process. Section 3.5 discusses how we obtain a global 3D heatmap and our final 3D object proposals.

### 3.3.1 Initialization with 2D Object Proposals

We collect $N$ video frames per scene using a RGB-D camera. Every $i^{th}$ video frame consists of a color image $I_i$, depth image $Z_i$, and pose of the camera $P_i$, using a depth based SLAM method such as Dense Visual SLAM [55]. The camera pose contains the rotation and translation measurements of the camera in world coordinate frame of reference: $P_i = [R_i, t_i]$. The RGB-D camera is assumed to be pre-calibrated. The camera intrinsic parameters - the focal length in $x$ and $y$ directions and optical center - are denoted by $f_x, f_y, [c_x, c_y]$, respectively. Together, these are represented by the intrinsic calibration matrix $K$:

$$
K = \begin{bmatrix}
    f_x & 0 & c_x \\
    0 & f_y & c_y \\
    0 & 0 & 1
\end{bmatrix}
$$  \hspace{1cm} (3.1)

We use a generic 2D object proposal technique such as edge-boxes [42] to obtain $M$ number of 2D object proposals per frame. Edge-boxes [42] provide a $5 \times 1$ measurement vector per bounding box:

$$
e_i^j = [x, y, w, h, c]_i^j \text{, } j \in 1, \ldots, M, \quad i \in 1, \ldots, N
$$  \hspace{1cm} (3.2)
Here, $[x, y]$ denote the top-left pixel coordinate of the bounding box, and $w, h$ refer to the width and the height in $x$ and $y$ directions respectively. $c$ corresponds to the confidence value of the bounding box which is proportional to the likelihood of that bounding box containing an object in it. A few top ranked (after non-maximum suppression) 2D object proposals are shown in Fig. 3.3. These 2D object proposals per image are treated as an input for our framework.

### 3.4 Depth-Based 2D Object Proposal Filtering

In this section, we describe how we use depth information per frame to improve current existing 2D object proposal techniques. As we only look at a single frame in this section, we drop the subscript $i$ for the sake of clarity for the single view problem formulation.

![Figure 3.3: Top ranked object proposals are displayed for images using Non-Maximum Suppression (best viewed in color). Object proposals are displayed in order of their confidence. Object proposal with highest confidence is displayed first. Any object proposals that share more than 10% region, i.e. intersection over union (IoU) greater than 0.1, are not shown. Using a simple NMS approach fails to select the best object proposals.](image)

3.4.1 2D Weighted Heatmap for Pixelwise Objectness

One problem with using object proposals is the redundancy of the majority of the proposals. The object proposal techniques are usually designed for high recall and Jaccard index (IoU). This results in several object proposals in image areas where an object might be located. Object recognition techniques are used on each
of these bounding boxes to detect and identify whether they contain an object. We ask a question: Is it possible to quickly improve or reject some of these proposals using scene geometry without performing expensive object detection and recognition techniques?

To select the best possible proposal, techniques similar to non-maximum suppression (NMS) are generally used. Object proposals with highest confidence locally are chosen, while any neighboring proposals that share a common region with the locally chosen proposal are rejected. However, there is no guarantee that the object proposals with higher confidence will actually be the best fitting object proposals for a given scene as shown in Fig. 3.3. Reference [3] also observed that object proposal techniques often fail to find good 2D boundaries of objects present in a video scene due to motion blur. For example, none of the top 1,000 proposals in Fig. 3.3(a) resulted in a good bounding box for the yellow bowl (IoU > 0.5). Our technique described in this chapter is able to overcome these motion blur based problems and find good 3D and 2D object proposals in such cases as well.

Thus, instead of deciding right away the best 2D proposals in a given scene, we use a weighted approach to create a heatmap. A heatmap is a two-dimensional real-valued matrix indicating the likelihood of each image pixel \([u, v]\) to be occupied by an object. We consolidate the confidence of each pixel by summing over all the object proposals \(\{e^j = [x^j, y^j, w^j, h^j, c^j]\}_{j=1}^M\) for a given image. We denote this heatmap as “baseline heatmap” \(\hat{H}\), and it is obtained by:

\[
\hat{H}[u, v] = \sum_{j=1}^M c^j \delta^j_o[u, v]
\]  

(3.3)

\[
\delta^j_o[u, v] = \begin{cases} 
1 & \text{if } u \in [x^j, x^j + w^j], \ v \in [y^j, y^j + h^j] \\
0 & \text{otherwise}
\end{cases}
\]  

(3.4)

where \(c^j\) denotes the confidence of the \(j^{th}\) object proposal, and \(\delta^j_o[u, v]\) is a binary filter that checks if the pixel \([u, v]\) is contained in the current object proposal \(e^j\). Here, \(c^j\) denotes the confidence of the \(j^{th}\) object proposal, and \(\delta^j_o[u, v]\) is a binary filter that checks if the pixel \([u, v]\) is contained in the current object proposal \(e^j\).

There are various advantages of using a heatmap based approach. Firstly, it is computationally fast and extremely simple to implement. The baseline heatmap can be computed in constant time using integral images [56], regardless of the size...
Figure 3.4: (a) Color image. (d) Corresponding depth image. (b) Baseline heatmap, $\hat{H}$. (e) Our heatmap, $H_{2D}$. (c) Baseline heatmap overlaid on color image. (f) Our heatmap overlaid on color image. We show a comparison between the baseline heatmap $\hat{H}$, and our improved heatmap $H_{2D}$. Our depth-assisted heatmap is able to filter out the background from the objects of interest using depth based statistics for every object proposal.

of bounding boxes. Secondly, the heatmap provides a much more fine-grained, per-pixel objectness map rather than an array of bounding boxes per frame. This makes it easier to transfer this information in 3D global heatmap. Instead of matching bounding boxes across video frames one by one, we project the entire heatmap and consolidate the confidence values of the 3D points in the global world coordinate frame of reference. Thirdly, the heatmap is an accumulation of statistics computed from all bounding boxes. It turns out that sometimes no bounding box may cover an object properly. In such cases, the heatmap obtained from all these bounding boxes may result in a far more intelligent boundary of the objects location in the scene. Thus, instead of matching bounding boxes across video frames one by one, we project the entire heatmap and consolidate the confidence values of the 3D points in the global world coordinate frame of reference.

The baseline heatmap provides a good idea where the objects of interest might be located in the scene. However, the baseline heatmap is still rather coarse as seen in Fig. 3.4(c). This is because object proposals tend to enclose a lot of non-object regions such as texture on the wall and floor or sides of the table as seen
in Fig. 3.4(a). To resolve this issue, we perform a depth based two-step filtering process to refine these initial 2D proposals: soft filtering and hard filtering.

3.4.2 Soft Filtering: Background Masking in Object Proposals

Soft filtering refers to making a binary decision for each pixel in an object proposal. We classify every pixel of an object proposal region into a foreground or background pixel by performing a quick segmentation using depth information. Let \( Z_{\text{min}} \) and \( Z_{\text{max}} \) be the minimum and maximum depth range of the current object region. We compute its mean: \( Z_\mu = \frac{Z_{\text{min}} + Z_{\text{max}}}{2} \). As our system is geared towards online application, we cannot afford complex segmentation techniques such as mean-shift or K-means. The object proposal is considered as containing background parts if the difference \( \Delta Z = Z_{\text{max}} - Z_{\text{min}} \) is above a certain threshold \( \epsilon_{\Delta} \). We trained for this parameter on the Object Segmentation dataset [57] and found \( \epsilon_{\Delta} = 0.5 \) m to give the best results. For such object proposals containing background portions, a pixel \([u, v]\) is classified as a foreground pixel if the pixel’s depth value \( Z[u, v] \) is less than \( Z_\mu \) and vice versa as given in Eq. 3.5. The background pixels in each bounding box proposal are assigned zero confidence value while the confidence value for foreground pixels is retained.

\[
\delta_s^j[u, v] = \begin{cases} 
1 & \text{if } Z[u, v] \leq Z_\mu^j \text{ or } \Delta Z_j^j < \epsilon_{\Delta} \\
0 & \text{otherwise}
\end{cases}
\]  

(3.5)

Our threshold is very conservative and only masks off background pixels when there is a high probability for an object proposal to contain the background. Figure 3.5 demonstrates the effect of this soft filtering method. The blue and magenta bounding boxes represent the regions that are detected to contain background parts and undergo background masking. The red and green bounding boxes represent regions with \( \Delta Z < \epsilon_{\Delta} \) and do not undergo this soft filtering process as they do not contain any significant background.

3.4.3 Hard Filtering: Culling Odd-Sized Object Proposals

Hard filtering refers to making a binary decision, accept or reject, for every object proposal of an image. The 2D bounding boxes given by various object proposal
Figure 3.5: Depth based filtering of 2D object proposals (best viewed in color). We only display a handful of object proposals for clarity purposes. In the soft filtering process, the blue and magenta bounding boxes are detected to contain background and undergo depth based background suppression while the red and green bounding boxes are detected to contain no major background. After background masking process, we check the actual size of each bounding box using depth information. The red and magenta bounding boxes are identified as areas that are either too small or too big in size or do not contain depth information (holes in depth image). These 2D object proposals are discarded. The blue regions, after background suppression, and green regions are detected to potentially contain good information and are accepted as valid object proposals.

Techniques can vary from anywhere between $2 \times 2$ pixels to spanning the entire image. To reduce the number of irrelevant bounding boxes, some approaches such as [3] ignore any small bounding boxes less than $20 \times 20$ pixels or any relatively large bounding boxes. However, sometimes this will result in losing important information in scenarios when the object of interest is located at a distance from the camera or when the camera is zoomed onto an object of interest momentarily.

To overcome this problem, we compute the size statistics of each bounding box given as an object proposal using the depth information available. An object proposal is discarded if it is estimated to be outside the desired range as follows:

$$
\delta_{h,j}(.) = \begin{cases} 
0 & \text{if } \frac{w^j \cdot Z^j_\mu}{f_x}, \frac{h^j \cdot Z^j_\mu}{f_y} < \epsilon_{\text{min}} \text{ or } \frac{w^j \cdot Z^j_\mu}{f_x}, \frac{h^j \cdot Z^j_\mu}{f_y} > \epsilon_{\text{max}} \\
1 & \text{otherwise}
\end{cases} (3.6)
$$

where $[\epsilon_{\text{min}}, \epsilon_{\text{max}}] = [2\text{cm}, 1\text{m}]$, and $w^j, h^j, Z^j_\mu$ correspond to width, height and mean depth of the foreground object in the $j^{th}$ object proposal. $f_x$ and $f_y$ are the
known focal length parameters in the \( x \) and \( y \) direction. Specifically, we reject any object proposal whose approximate cross-section size is bigger than 1 m \( \times \) 1 m or smaller than 2 cm \( \times \) 2 cm. Note that an object proposal of size 3 m \( \times \) 1 cm, such as a long stick, is still considered to be a desirable object and accepted as a valid object proposal. The parameters \( \epsilon_{\min} \) and \( \epsilon_{\max} \) are empirically set in this discussion, but they can be adapted to best fit different RGB-D sensing cases. We also reject the object proposals that contain no depth information. This usually happens when the regions inside these bounding boxes are too far from the depth camera. In Fig. 3.5, the red and magenta bounding boxes represent object proposals that are rejected based on our hard-filtering process. Meanwhile, the green and blue bounding boxes represent object proposals that are accepted as valid object proposals.

Thereafter, the confidence for each object proposal window is updated pixel by pixel and our improved weighted 2D heatmap is computed for every pixel \([u, v]\) as follows:

\[
H_{2D}[u, v] = \sum_{j=1}^{M} \delta_i^j(\cdot) \cdot \delta^j_s[u, v] \cdot c^j_o[u, v]
\] (3.7)

In summary, our depth-based filtering approach reduces the computation required and improves the precision of our object proposals as the irrelevant object proposals are filtered out. A baseline heatmap (without the depth based filtering process) and our weighted heatmap are shown in Fig. 3.4(c, d). The objects of interest - cup and caps - stand out in our heatmap. Additionally, the background is correctly detected and assigned low confidence as shown in Fig. 3.4(f).

### 3.5 Multi-view Fusion and Refinement for a Global 3D Heatmap

In this section we describe our second main contribution of how we fuse a sequence of weighted 2D heatmaps \( H_{2D} \) in 3D, using depth information and camera pose efficiently.

Let \( \mathbf{x}_w = [x, y, z]^T \) be a 3D point in the world coordinate frame. It can be projected onto a 2D camera plane, and we denote the projected 2D point by \( \mathbf{x}_c = [u, v]^T \) and compute it as

\[
\begin{bmatrix}
\mathbf{x}_c \\
1
\end{bmatrix} = \lambda \mathbf{K} \mathbf{P} \begin{bmatrix}
\mathbf{x}_w \\
1
\end{bmatrix}
\]

Here \( \lambda \) is a proportionality constant, while \( \mathbf{K} \) and \( \mathbf{P} \), as defined earlier, denote the intrinsic calibration
matrix and the camera pose, respectively. For notational convenience, we represent this projection onto the camera plane as a function $\pi$:

$$x_c = \pi(K, P, x_w)$$

(3.8)

Similarly, a 2D pixel $x_c$ can be projected and transformed onto the world coordinate frame by using its depth value $Z[x_c]$:

$$x_p = \begin{bmatrix}
\frac{Z[x_c]}{f_x}(u - c_x) \\
\frac{Z[x_c]}{f_y}(v - c_y) \\
Z[x_c]
\end{bmatrix}$$

(3.9)

$$x_w = R^T x_p - R^T t$$

(3.10)

where $x_p$ is the 3D point in the camera frame of reference. It is then transformed to the world coordinate frame using the camera pose $P$. We define the projection and transformation as a function $\pi^{-1}$:

$$x_w = \pi^{-1}(K, P, x_c, Z[x_c])$$

(3.11)

### 3.5.1 Horizontal Supporting Plane Removal

After the weighted 2D heatmap $H_{2D}$ is computed for the current frame, the image pixels are projected onto 3D using Eq. 3.11. The 3D points are registered with the global point cloud $H_{3D} \in \mathbb{R}^{N \times 8} : [x, y, z, r, g, b, c, f]$. The first three components $[x, y, z]$ represent the global 3D location of the point $x_w$. The next three components $[r, g, b]$ represent its color. $c$ records the consolidated heat value (confidence) of the 3D point. Finally, $f$ denotes the number of times (frequency) that a point has been seen in RGB-D frames. We initialize the 3D heatmap $H_{3D}$ for the first frame with unit frequency for all the valid image pixels.

Our next task is to estimate dominant horizontal supporting planes, with the motivation that they are usually not objects of interest in a given scene. Given three 3D points $\{x_1, x_2, x_3\}$, a 3D plane $p = [n_x, n_y, n_z, b]^T$ passing through
these three points is computed as follows:

\[
\mathbf{n} = \begin{bmatrix}
    n_x \\
    n_y \\
    n_z
\end{bmatrix} = \frac{(x_2 - x_1) \times (x_3 - x_1)}{||(x_2 - x_1) \times (x_3 - x_1)||_2}
\]

\[
b = -\mathbf{n}^\top x_1
\]

A 3D point, \(x_1\), is considered to be lying on the plane \(p\) if

\[
|p^\top \cdot \begin{bmatrix} x_i \\ 1 \end{bmatrix}| < \epsilon_p
\]

(3.12)

where \(\epsilon_p\) is a small threshold to account for noisy data. We found \(\epsilon_p = 0.005\) empirically to work best. The 3D points that lie on this plane are considered inliers and the points that exceed the threshold are considered outliers. We denote the inliers for the \(j^{th}\) detected plane \(p^j\) as \(S^j\). We use a RANSAC based technique where we select three neighboring points in a given frame to estimate a plane passing through these points and then compute inliers and outliers for the plane using Eq. 3.12. This process is repeated 10,000 times to find the top five distinct dominant planes. The top planes for a sample frame in the order of decreasing number of inliers are shown in Fig. 3.6(c-f).

To improve the object localization, we aim to separate the objects of interest from a plane that may be present in the top ranked points. As shown in Fig. 3.6(b), objects of interest often lie on a supporting plane such as the table underneath the objects. This supporting plane is often contained in the 2D object proposals even after our depth based filtering process. Since a scene capture can start with a camera angle in any direction, we cannot make any underlying assumptions about the direction and location of the horizontal planes in a given scene. Here we assume that our dataset contains one supporting plane. However, it can be easily extended to remove more than one supporting plane. We observed that using the RANSAC based technique to find the plane with most inliers often resulted in selecting the horizontal plane containing the floor or vertical plane containing room walls. Due to complicated indoor lighting conditions, the table color may also vary significantly from pixel to pixel. Hence, using a color based technique may not always be successful in finding our plane of interest. In addition, it is challenging to know how many dominant redundant planes there might be in a
Figure 3.6: Our plane removal process takes into account the heatmap of the scene. A sample 3D point cloud and its weighted heatmap are shown in (a) and (b) respectively. We display the most dominant planes in order of number of inliers in (c)-(f). The plane in (f) is selected for plane removal as its heatmap based confidence is highest among top ranked planes.

given scene. Thus, removing all the dominant planes is also not an ideal solution.

Since we are only interested in removing the supporting plane underneath objects of interest, we utilize the obtained 2D heatmap to our advantage. Based on the inlier criterion defined in Eq. 3.12, we estimate the plane of interest for the first image frame by selecting the plane $p_j^*$ that has the highest accumulated heat value as follows:

$$j^* = \arg \max_j \sum_i \left( H_{2D}(x_i) : \forall x_i \in S^j \right)$$

This heatmap based plane estimation assists us in consistently finding the correct plane of interest as shown in Fig. 3.6. We assign zero confidence to all the pixels lying on the plane of interest. For the next frame, we choose to use the camera pose to project this plane, and compute the planar points by projecting onto the camera plane without needing to recompute the plane again. The pixels corresponding to the plane are assigned zero confidence to obtain a filtered 2D heatmap.
Figure 3.7: (a) Color image. (b) Matched pixels. (c) Weighted 2D heatmap, $H_{2D}$. (d) Heatmap after plane removal, $\tilde{H}_{2D}$. (e) Global heatmap, $H_{3D}$ projected onto image plane. Left column displays different RGB frames from table_1 scene in [5]. 2nd columns shows the matched pixels of current frame with the previous frame. 3rd column highlights our weighted 2D heatmap after depth based filtering. 4th column displays our refined heatmap after plane removal. Finally, the 5th column shows the current global heatmap of the entire scene. The global heatmap has been scaled for visualization.

\[ \tilde{H}_{2D} : \]
\[
\tilde{H}_{2D}[u,v] = \begin{cases} 0 & \forall [u,v] \in \{ \pi(K, P, x_i) | x_i \in S^j \} \\ H_{2D}[u,v] & \text{otherwise} \end{cases}
\]

We recompute the plane parameters $p^{i*}$ every ten frames to account for camera drift that SLAM methods often suffer from. We also store the plane parameters in a separate matrix $P^*$ to be used later. Instead of repeating the entire plane estimation process, we utilize our rough knowledge of the plane location to compute the new plane of interest quickly and efficiently. We are able to consistently remove the correct supporting plane using this automated plane removal approach. One situation where this approach fails is where a camera trajectory starts with looking at the floor and slowly pans toward the table. The floor is selected as the plane of interest in initial frames when the table and other objects are not seen in the cam-
Figure 3.8: We show our 3D heatmap generation process using image warping. We initialize our global heatmap, $H_{3D}$, after first frame. Once we obtain the current frame, we warp the previous image on top of the current image and find the matching points. The matching points are shown in various unique colors and the unmatched points are shown in gray. We consolidate these points’ confidence value by adding their individual confidence. We update their 3D location in world coordinate frame and color information by averaging their individual measurements. The unmatched points (shown in grey) are added to the global heatmap with their respective confidence values and unit frequency.

era’s field of view. Once the table occupies enough of the camera’s view that its accumulated confidence is higher than the floor, the table is from thereon selected as the plane of interest. In these intermediate frames the partially seen table is not removed and thus retains its high confidence value in the heatmap. We discuss how to resolve this issue in Section 3.5.4. We refer to the frames where we recompute plane parameters as keyframes. Unlike SLAM methods where keyframes are used to estimate the camera pose of the current frame, we only use the keyframes to project the plane onto the current frame, and assign the corresponding planar pixels zero confidence as shown in Fig. 3.7(d).

### 3.5.2 3D Heatmap Generation via Multi-view Fusion

After we compute the filtered 2D heatmap $\hat{H}_{2D}$, and use the corresponding depth information available to project the points in 3D, the next step is to fuse this information with the existing 3D heatmap $H_{3D}$.

A standard approach to fuse this information with the current 3D heatmap is to
find the closest points using techniques such as Iterative Closest Point (ICP) [58]. However, such an approach quickly becomes computationally expensive, which deviates from our goal of developing a fast and real-time approach for 3D object proposals. Another way to tackle this problem is to use the poses estimated by depth based SLAM methods to allocate points in 3D voxels or Octrees. However, due to the sheer amount of 3D points and span of the room, such an approach also requires a large amount of memory and incurs heavy computation loads. We resolve this issue by using image warping and creating a 2D index table. First, we initialize the mapping from 2D heatmap to 3D heatmap for the first frame. This bijective mapping, called indexMap, stores the location of each pixel of current 3D heatmap in the global $H_{3D}$ heatmap.

Using Eq. 3.15, we utilize the previous frame’s depth information to warp the image onto current $i^{th}$ frame. We round the project pixel location to nearest integer and compare their depth and color values per pixel. Let us assume that pixel $\tilde{x}_c$ of previous frame is warped onto $x_c$ of the current frame.

$$x_c = \pi(K, P^i, P^{(i-1)}, \tilde{x}_c, Z^{(i-1)}[\tilde{x}_c])$$  \hspace{1cm} (3.15)

Based on this information, we compute the difference in intensity and depth at each matched pixel:

$$\Delta I^i[x_c] = ||I^i[x_c] - I^{(i-1)}[\tilde{x}_c]||_2$$  \hspace{1cm} (3.16)

$$\Delta Z^i(x_c) = |Z^i[x_c] - \tilde{Z}^{(i-1)}[\tilde{x}_c]|$$  \hspace{1cm} (3.17)

where $\tilde{Z}^{(i-1)}[\tilde{x}_c]$ denotes the warped depth value in current frame. If the projected pixel’s color and depth information is within a certain threshold ($\epsilon_I, \epsilon_Z$) of the current pixel’s information, the two corresponding pixels are considered a true match and the index is copied to the current pixel. In the case where more than one pixel from the previous frame matches a pixel in the current frame, then the pixel corresponding to lower warped depth (foreground) is chosen as the matching pixel. This matching is shown in Fig. 3.7(b). If the color and depth measurements are significantly different or if no pixel from the previous frame is warped onto current pixel, the pixel is identified as a new point and added to the global 3D heatmap. A new index is created for such pixels and added to the current indexMap. Since this approach requires depth information at each pixel, if the depth information is not available (holes in depth image) or contains noisy depth, this can lead to
wrongly matched pixels. We choose to ignore these pixels as our primary goal is to obtain fast and efficient 3D proposals and performing this step for every image can get expensive. The update for matching points and global heatmap update is shown in Fig. 3.8. Nevertheless, if more robust results are required, one can fill the holes using a simple nearest-neighbor method or an adaptive filter using the corresponding color images [59].

For all matched points, the original 3D point’s confidence value is increased by the current matched pixel’s confidence value and the counter is incremented by a unit. In addition, the color and location of the 3D points are adjusted by weighted average of the current 3D location of the pixel and the global location of the matched point. This accounts for any minor drift error that might occur while estimating the camera trajectory in an indoor environment.

3.5.3 3D Heatmap Filtering using Average Confidence Measure

Once we obtain our weighted 3D heatmap, any points that are seen less than five times or 5% of the total number of frames, whichever is lower, are identified as unwanted points and discarded. We use a metric: pseudo-average confidence, \( p_{\mu} \), to rank the global 3D points. The pseudo-average confidence of the \( i^{th} \) point is computed as:

\[
\bar{c}(i) = \frac{H_{3D}(i; c)}{H_{3D}(i; f)} + \tau, \quad i = 1, \ldots, N
\]  

(3.18)

where \( H_{3D}(i; c) \) denotes retrieving the \( c \) (confidence) element of the \( i^{th} \) point stored in the 3D heatmap \( H_{3D} \), similarly for \( H_{3D}(i; f) \) that returns the \( f \) (frequency) element.

Pseudo-average ranks points seen more often slightly higher than points seen less often. Intuitively, this makes sense as we should see the 3D points lying on objects of interest more often than other points. We compute the pseudo-average heat value for all the 3D points and retain only those points that have \( \bar{c}(i) \geq \epsilon \).

This ensures that we obtain good precision while maintaining an acceptable recall value. Depending on the situation, if we want better precision or recall we can increase or decrease this threshold ratio respectively. An example of the top ranked 3D point cloud is shown in Fig. 3.9(a).
3.5.4 3D Point Clustering and 3D Bounding Box Generation

As discussed in Section 3.5.1, it is possible that the scene capture starts without the objects and table in the camera’s field of view. The scene capture may start with room floor or walls in its entire field of view and then move to the area of interest. In cases where the table is only partially seen (a few pixels) when the camera field of view is moving towards the objects of interest, the accumulated heatmap value of the planes corresponding to walls and floors may be temporarily higher in a few image frames. In such cases, the floor and walls will be removed from the heatmap but the table will be left untouched. Thus, some outer regions of the table may still be present in our final ranked points as seen in Fig. 3.9(a). We perform a final plane removal using the plane parameters stored in the matrix \( P^* \) obtained from our previous keyframes. The plane parameters for each keyframe may correspond to the floor or vertical wall when the table is not seen. Thus, we first identify different planes that were removed using k-means (walls, rooms, table etc.). For each unique plane, we find the best fit plane using these different plane parameters. Thereafter, we perform a final plane removal step on our top ranked points by finding any points that satisfy Eq. 3.12 for the estimated plane. The final filtered 3D point cloud is shown in Fig. 3.9(b).

After plane removal, we perform density based clustering using density-based spatial clustering of applications with noise (DBSCAN) [60] on our filtered top ranked points. DBSCAN groups together points that are closely packed together and marks points as outliers that lie alone in low-density regions. This allows us to reject any points that may belong to uninteresting regions such as walls and table that may have been ranked in our top filtered points. The advantage of using a density based approach over other techniques such as k-means is that we do not need to specify the number of clusters that are present in our filtered data. Thus, depending on the scene complexity, this approach can automatically select the relevant number of regions of interest in the scene. DBSCAN can run in an overall average runtime complexity of \( O(n \log(n)) \). An example of DBSCAN based clustering is shown in Fig. 3.9(c).

After density based clustering, we estimate a tight 3D bounding box for each cluster. However, finding a good bounding box is challenging in camera’s frame of reference since the horizontal direction (normal to the floor) may not be the same as the camera’s horizontal direction. First we estimate the direction of gravity (normal) by using the entire 3D point cloud obtained after our data collection
as explained in Section 3.5.1. Once the normal $n_c$ to the horizontal plane in the camera’s coordinate frame of reference is obtained, each cluster is transformed to the world coordinate frame where the orthogonal directions X, Y, and Z match the standard normal vectors - $[1; 0; 0; ]$, $[0; 1; 0]$, and $[0; 0; 1]$ respectively using Eq. 3.19. Thereafter, we find the minima and maxima in each of the three orthogonal directions and draw a bounding box around the clusters. This bounding box is then transformed back to the original coordinate frame.

$$
v = n_c \times n_w$$

$$s = ||v||_2$$

$$c = n_c \cdot n_w$$

$$R = I + [v]_x^2 + \frac{(1-c)}{s^2}[v]_x^2$$ (3.19)

where $n_c$ is the normal to the detected horizontal plane in the camera’s coordinate frame, $n_w = [0; 1; 0]$ is the world coordinate’s normal direction. $[v]_x$ is the skew symmetric matrix of vector $v$. If the bounding boxes of neighboring clusters intersect, we combine those clusters. This usually happens when an object breaks into two sub-parts due to pose estimation errors or in the presence of specular objects such as a soda can. We reject any bounding boxes that have a volume of less than 1 cm$^3$ as these small patches are usually a part of the walls or floor plane. The final refined 3D point cloud with the respective bounding boxes is shown in Fig. 3.9(d). This modular approach assists in finding tight bounding boxes in the direction of gravity for each object of interest.
3.6 Experimental Evaluation

We use four datasets to conduct our experimental analysis and evaluation - Object Segmentation Dataset [57], UW-RGBD Scene dataset [2], RGBD scenes dataset [5], and our own dataset.

The Object Segmentation Dataset (OSD) consists of 111 labelled RGBD images in six subsets. The images vary from having two small objects on a table to containing more than ten objects stacked side by side and on top of each other. The authors provide pixelwise labeling of each instance of the objects. We use these labelled images to create 2D groundtruth bounding boxes for each object present in the dataset to enable learning and for Precision-Recall evaluation. The depth images are pre-aligned with the color images. We primarily used this dataset to train our parameters.

The UW-RGBD Scene Dataset contains 14 scenes reconstructed from RGB-D video sequences containing furniture and some table-top objects such as caps, cereal boxes and coffee mugs. The scenes contain depth and color frames from a video collected by moving around the scene. The dataset provides a globally labeled 3D point cloud. We used Dense Visual SLAM [55] to obtain the camera pose per frame to fuse the frames together. Our 3D point cloud is slightly misaligned with the groundtruth point cloud by a few mm. It is not trivial to align these to point clouds due to different error characteristics of the point clouds. Thus, for our current evaluation, we decided to ignore this mismatch as the misalignment is marginal. We primarily use this dataset for our 3D evaluation and comparison with existing state-of-the-art techniques in 2D by projecting our bounding boxes on the image plane.

RGBD scenes consist of eight realistic environment scenes of a lab, kitchen and office room. The objects of interest are placed on kitchen slabs and tables in a room. The authors provide groundtruth 2D bounding boxes for various objects of interest such as soda can, cap, and flashlight. However, some objects such as laptops, computer mice and other kitchen equipment are labelled as background for this dataset. The authors used depth and color images to segment and identify the objects present in each frame independently. We use this dataset to show our results in a more cluttered environment and compare our results in 2D by projecting our bounding boxes on image plane and comparing with the 2D groundtruth bounding boxes.

The existing RGBD datasets are intended for benchmarking category indepen-
dent object segmentation and identification purposes, and thus only provide limited test cases. The effect of error in the SLAM algorithm is also ignored. Therefore, we collected our own sequence with more challenging situations where the scene is crowded and the objects are placed in proximity. We also consider imperfect SLAM and demonstrate that our method is only marginally affected given inaccurate camera poses estimation. We also collected our own dataset to highlight our method’s results when the scene is crowded or when objects are in proximity to each other where an intensity based SLAM method may not be able to provide a good semi-dense depth map.

3.6.1 Training on OSD and Comparison with Edge-boxes

**OSD:** We first evaluate results on Object Segmentation Dataset (OSD) [57] to demonstrate how our depth based filtering improves on edge-boxes. We use this dataset to train our parameter $\epsilon_\Delta$ in Eq. 3.5. We used 1000 object proposals per image on training subset and found $\epsilon_\Delta = 0.5$ m to give the best results on test dataset as well. Using depth based 2D object proposal filtering as described in Section 3.4, we are successful in rejecting 5.1% of the object proposals provided by edge-boxes across 111 images.

From here on, the different parameters discussed in Sect. 3.4 and 3.5 are as follows:

$$\{\epsilon_\Delta, \epsilon_{\text{min}}, \epsilon_{\text{max}}, \tau\} = [0.5m, 2cm, 1m, 10]$$
$$\{\epsilon_p, \epsilon_I, \epsilon_Z, \epsilon\} = [0.005, 0.05, 0.01, 0.25]$$

**UW-RGBD Scene Dataset:** We also computed the acceptance and rejection rate on UW-RGBD dataset for our depth based filtering process. Based on the background masking and culling odd sized proposals as described in Sections 3.4.2 and 3.4.3, 7% of the object proposals are rejected, 31.3% are fully accepted and 61.7% of the object proposals undergo partial filtering to mask the background on the entire dataset.

**RGBD Scenes Dataset:** We use the groundtruth 2D bounding boxes provided in RGYD Scenes dataset to report the Average Precision, Recall and Success rate. As we treat each frame independently, we do not perform multi-view fusion as described in Section 3.5. Instead, after depth based filtering, we re-
move the unwanted supporting plane in 3D (Section 3.5.1), filter the 3D heatmap (Section 3.5.3) and cluster the 3D points to compute 3D bounding boxes (Section 3.5.4). We project the 3D points inside each of the 3D bounding boxes back to the image plane and compute the 2D bounding boxes around these pixels. As noted earlier, most 2D object proposal techniques aim for a high recall. Since our goal is fast and precise 3D object proposals, we obtain 5-20 object proposals depending on the scene complexity.

Let $BB_g(i)$ be the $i^{th}$ groundtruth 2D object proposal, $BB_e(j)$ and $BB_o(j)$ be the $j^{th}$ edge-boxes and our 2D object proposals. Let $M_e$ and $M_o$ be the total number of object proposals computed by edge-boxes and our method. First, we use the standard definition of IoU:

$$
\text{IoU}(i) = \max_j \left( \frac{BB_g(i) \cap BB_o(j)}{BB_g(i) \cup BB_o(j)} \right) \quad \forall j \in 1, \ldots, M \quad \forall i \in 1, \ldots, N \tag{3.20}
$$

where $M$ and $N$ are the total number of output object proposals and groundtruth object proposals respectively in a given scene. Detection rate (DR), sometimes also referred to as Average Recall, is defined as:

$$
DR = \frac{1}{N \cdot K} \sum_{k=1}^{K} \sum_{i=1}^{N} (\text{IoU}(i; k) \geq 0.5) \tag{3.21}
$$

where $K$ is the total number of scenes in the dataset and $\text{IoU}(i; k)$ refers to the IoU for $i^{th}$ groundtruth object proposal in $k^{th}$ scene. We define a modified Intersection over Union ($\text{IoU}_o$) for $j^{th}$ object proposal as follows:

$$
\text{IoU}_o(j) = \max_i \left( \frac{BB_g(i) \cap BB_o(j)}{BB_g(i) \cup BB_o(j)} \right) \quad \forall j \in 1, \ldots, M \quad \forall i \in 1, \ldots, N \tag{3.22}
$$

The difference between our modified IoU and standard IoU is that we estimate the best intersection per output object proposal while the standard IoU is computed per groundtruth object proposal. Our definition heavily penalizes any redundant object proposals. We obtain zero IoU for object proposals that do not intersect with ground truth object proposals. Based on this definition, similar to Eq. 3.21, we define Success Rate ($SR$) as:

$$
SR = \frac{1}{M \cdot K} \sum_{k=1}^{K} \sum_{j=1}^{M} (\text{IoU}_o(j; k) \geq 0.5) \tag{3.23}
$$
Figure 3.10: Final results for a sample scene in UW-RGBD dataset for our approach after removing one component. We obtain results in (a) without depth based filtering as described in Section 3.4. Results in (b) and (c) are obtained when background masking and bounding box rejection are not performed respectively. (d) shows results without plane removal. (e) shows our result when all of these components - background masking, bounding box rejection, and plane removal - are used together.

where $K$ is the total number of scenes in the dataset and $IoU_o(j; k)$ refers to our modified IoU for the $j^{th}$ object proposal on the $k^{th}$ scene. Success rate can also be interpreted as signal-to-noise ratio (STN) and has been used previously in [61] for 2D object proposal analysis. We use different numbers of input 2D edge-boxes proposals varying from 50 to 2,000 and report our results in Table 3.1.

On average we obtained 6.57 output 2D object proposals for our method. As shown in the Table 3.1, using scene geometry enables us to significantly improve on edge-boxes. We show remarkable improvement in success rate. Our success rate

<table>
<thead>
<tr>
<th>num proposals</th>
<th>EB</th>
<th>Our s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SR</td>
<td>DR</td>
</tr>
<tr>
<td>50</td>
<td>0.21</td>
<td>0.72</td>
</tr>
<tr>
<td>100</td>
<td>0.16</td>
<td>0.78</td>
</tr>
<tr>
<td>500</td>
<td>0.08</td>
<td>0.88</td>
</tr>
<tr>
<td>1000</td>
<td>0.05</td>
<td>0.89</td>
</tr>
<tr>
<td>2000</td>
<td>0.03</td>
<td>0.90</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison between edge-boxes and our technique per frame on UW-RGBD Scene dataset. SR and DR refer to success rate and detection rate respectively.
rate is stable regardless of number of input object proposals while the success rate reduces drastically for edge-boxes. This is because the object proposals are chosen in order of their confidence values. The more input object proposals, the less likely they are to contain an object of interest. Our detection rate is higher than edge-boxes when using fewer proposals. It is slightly less than edge-boxes when using 1000 or more proposals. This is due to occluded objects present in the scene. Since this analysis is frame independent, our method only looks at the current frame and finds tight bounding boxes around objects seen in the frame currently. With more input proposals, edge-boxes is able to propose 2D bounding boxes for these fully or partially occluded objects, which increases its detection rate. We are able to overcome this issue by leveraging multi-view information as we are not concerned about per view detection rate.

3.6.2 Multi-view

**UW-RGBD Dataset**: After we obtain 3D heatmap per frame, we use multi-view information to fuse this information. We use the pose estimated by Dense Visual SLAM [55]. It uses depth images along with color information to create a globally consistent 3D map and outputs a camera pose per frame. However, like other real-time SLAM algorithms that do not perform expensive optimization of camera pose and 3D point clouds, it also suffers from camera drift errors. Our algorithm is designed such that it is tolerant to small drift and noisy 3D point clouds. We perform object proposal at the same time as SLAM, and do not require the whole sequence to be captured first. Figure 3.12 displays a few examples of the output 3D point cloud, top ranked points and computed 3D object proposals of the scenes in the UW-RGBD dataset.

The entire process takes, on average, 3.03 seconds for VGA resolution in MATLAB. However, most of the time (> 2.0 s) is spent in accessing the storing and accessing the 3D point cloud. A detailed time evaluation of various techniques used in our approach is shown in Table 3.2. As we aim for a fast and efficient algorithm that is capable of online processing of 3D object proposals, we downsample the images by 2. This reduces the time taken per frame to less than one second in MATLAB on a single core CPU.

First, we show the average IoU, both in 2D and 3D, obtained per object of interest for the UW-RGBD Scene dataset to demonstrate the effectiveness of each
Table 3.2: Analysis of average run-time performance per frame of our approach. Our experiments were conducted on a single core Intel Xeon E5-1620 CPU.

<table>
<thead>
<tr>
<th>Method</th>
<th>VGA</th>
<th>↓ 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overall</strong></td>
<td>3.03 s</td>
<td>0.973 s</td>
</tr>
<tr>
<td>Edge-boxes [42]</td>
<td>0.251 s</td>
<td>0.078 s</td>
</tr>
<tr>
<td>Depth based Filtering</td>
<td>0.373 s</td>
<td>0.195 s</td>
</tr>
<tr>
<td>Plane removal</td>
<td>0.14 s</td>
<td>0.06 s</td>
</tr>
<tr>
<td>Global 3D heatmap (c,f)</td>
<td>0.506 s</td>
<td>0.132 s</td>
</tr>
<tr>
<td>Global 3D heatmap (X,C)</td>
<td>1.481 s</td>
<td>0.359 s</td>
</tr>
</tbody>
</table>

Table 3.3: Comparison of average IoU for various objects present in UW-RGBD dataset. The benefits of each of our contributions - background suppression, rejection of odd sized bounding boxes, and supporting plane removal are clearly visible.

<table>
<thead>
<tr>
<th>Method Used</th>
<th>2D</th>
<th>3D</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IoU</td>
<td>IoU_o</td>
<td>SR</td>
<td>DR</td>
<td>IoU</td>
<td>IoU_o</td>
<td>SR</td>
<td>DR</td>
<td>IoU</td>
<td>IoU_o</td>
<td>SR</td>
<td>DR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>noHmap</td>
<td>0.32</td>
<td>0.19</td>
<td>0.17</td>
<td>0.30</td>
<td>0.17</td>
<td>0.07</td>
<td>0.01</td>
<td>0.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>noSFilt</td>
<td>0.51</td>
<td>0.25</td>
<td>0.29</td>
<td>0.60</td>
<td>0.33</td>
<td>0.15</td>
<td>0.13</td>
<td>0.31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>noHFilt</td>
<td>0.77</td>
<td>0.47</td>
<td>0.60</td>
<td>1.00</td>
<td>0.53</td>
<td>0.32</td>
<td>0.35</td>
<td>0.53</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>noPlRem</td>
<td>0.13</td>
<td>0.10</td>
<td>0.02</td>
<td>0.02</td>
<td>0.03</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Our s</td>
<td>0.78</td>
<td>0.50</td>
<td>0.62</td>
<td>1.00</td>
<td>0.55</td>
<td>0.34</td>
<td>0.45</td>
<td>0.66</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

of our contributions in Table 3.3. We estimate the 2D bounding boxes by projecting the 3D points inside each bounding box onto the image plane and computing a 2D bounding box around it. We repeat this procedure to estimate groundtruth 2D bounding boxes in addition to using the groundtruth labeled point cloud. The left column in Table 3.3 represents one step that was skipped to estimate the 3D bounding boxes.

*noHmap* refers to our approach without using the weighted heatmap, $H_{2D}$. In this approach, we skipped both soft filtering and hard filtering as described in Section 3.4. *noSFilt* computes the results without performing background masking using depth information, and *noHFilt* refers to our approach without rejecting odd sized bounding boxes that are mainly part of the background. *noPlRem* computes our results without performing plane removal per frame as described in Section 3.5.1. Finally, *Our s* refers to our method that integrates all the steps together to compute the 3D object proposals. We observe that each step is vital in improving the accuracy of our algorithm. A good supporting plane estimation plays an important role in removing the table underneath the objects for tight bounding box estimation. We show the final top ranked points along with the estimated bounding boxes in Fig. 3.10.
We also compare our results with [3]. Our approach considers chairs as objects of interest. However, since [3] treat chairs as background objects we also leave them out of our analysis for a fair comparison in Fig. 3.11. Once we obtain the 3D bounding boxes, we compute all the points that lie inside the current bounding box and project them onto the current image plane. Thereafter, we estimate the 2D bounding box that surrounds these projected points. We repeat this procedure for the groundtruth bounding boxes and compute the standard IoU using Eq. 3.20. Our 2D proposals (with and without downsampling) consistently outperform [3]. 3D intersection is extremely sensitive to noise. A small misalignment of 12.6% in each of three orthogonal directions reduces the 3D IoU to below 0.5. While our recall rate is good for low 3D IoU, it dips below 0.5 quickly due to the small mismatch between our point cloud and the groundtruth point cloud.

Like [2], we use the groundtruth labeling per point to obtain Precision-Recall results for our 3D object proposals. Consider the groundtruth points labeled as soda can, bowl, cap, chair as points of interest ($\hat{P}$) and the background such as sofa, table, floor as redundant points ($\hat{N}$). We overlay our 3D bounding boxes on the groundtruth point cloud and check if a point of interest, $i$, lies inside or outside our proposals. We consider the points of interest that lie inside our object proposals as True Positives ($TP$) and the redundant points that lie inside our object proposals False Positives ($FP$):

$$TP = \sum_{i=1}^{N_g} (\hat{P}(i) \in BB_o(j)) \quad \forall j \in 1 \ldots M$$

$$FP = \sum_{i=1}^{N_g} (\hat{N}(i) \in BB_o(j)) \quad \forall j \in 1 \ldots M$$

where $N_g$ refers to the number of groundtruth points in a given scene. We compute the Average Precision, Average Recall and F-measure as:

$$AP = \frac{1}{K} \sum_{k=1}^{K} \left( \frac{TP_k}{TP_k + FP_k} \right)$$

$$AR = \frac{1}{K} \sum_{k=1}^{K} \left( \frac{TP_k}{P_k} \right)$$

$$FM = \frac{2 \times AP \times AR}{AP + AR}$$

56
Figure 3.11: Our 2D and 3D recall rate with varying threshold for IoU on UW-RGBD dataset. Our 2D proposals comfortably outperform [3]. Since we exploit scene geometry, downsampling images by 2 does not have a significant impact on our recall rate.

where $K$ is the total number of scenes in the dataset. $\hat{P}_k$, $TP_k$, and $FP_k$ refer to the total points of interest, true positives and false positives in $k^{th}$ scene. We repeat this process in 2D to obtain Precision-Recall measurements in 2D by projecting the bounding boxes in 2D first before finding the true positives and false positives.

We compare our object proposals with state-of-the-art segmentation and classification results reported on the UW-RGBD dataset in Table 3.4. Even for 3D precision-recall measurements, we obtain the best mean Average-Precision while maintaining an acceptable recall. The groundtruth labeling considers objects such as desktops as part of the background. This decreases our average precision as these detected objects are considered as False Positives. If we ignore the scenes 13 and 14 in the dataset where we see these objects, our average precision increases from 93.46 to 98.76. Another issue affecting our Recall performance is the presence of chairs. The camera’s field-of-view is concentrated mainly on the objects in the scene and hence the chairs (especially those under the table) are not seen frequently as they are filtered out by our technique. This reduces our recall rate if we consider chairs as objects of interest. We also compute our recall without chairs as objects of interest and are reported in Table 3.4. Our 3D recall is lower than [2] because of the small misalignment between our point cloud and groundtruth point cloud. This misalignment results in some points of interest ($\hat{P}$) that are not included in our true positives ($TP_k$) which results in an artificially lower recall rate.
Figure 3.12: Our 3D object proposals for UW-RGBD dataset (best seen in color). Left column shows the point cloud of the different scenes in UW-RGBD dataset. 2nd column shows our top ranked filtered points and finally 3rd column highlights the resulting 3D object proposals based on density based clustering.

Our technique is agnostic about the number of objects that may be present in the scene and hence, can scale well for busy scenes that may contain lots of objects. We obtain on average 6.57 3D object proposals per scene. Our results can be further improved with a better camera pose estimation as in some cases the objects break into two or more discontinuous point clusters due to noisy camera pose. This results in multiple distinct object proposals for one object, essentially dividing the object into two or more pieces.

**RGBD Scenes:** We use RGBD scenes as well to show our results in a more cluttered environment. RGBD scenes contain seven different scenes ranging from
Table 3.4: Analysis of our 3D object proposals on UW-RGBD dataset in comparison to [1], [2], and [3]. We achieve near run time performance if we downsample the data for our analysis. Our experiments were conducted on a single core Intel Xeon E5-1620 CPU.

<table>
<thead>
<tr>
<th>Method</th>
<th>Run-time</th>
<th>AP</th>
<th>AR</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>DetOnly [1]</td>
<td>1.8s</td>
<td>61.7</td>
<td>81.9</td>
<td>70.38</td>
</tr>
<tr>
<td>HMP2D+3D [2]</td>
<td>4.0s</td>
<td>92.8</td>
<td><strong>95.3</strong></td>
<td><strong>94.03</strong></td>
</tr>
<tr>
<td>Pillai [3]</td>
<td>1.6s</td>
<td>81.5</td>
<td>59.4</td>
<td>68.72</td>
</tr>
<tr>
<td>Ours</td>
<td>3.03s</td>
<td>93.46</td>
<td>76.19</td>
<td>83.95</td>
</tr>
<tr>
<td>Ours w/o chairs,desktop</td>
<td>3.03s</td>
<td><strong>98.76</strong></td>
<td>81.66</td>
<td>89.40</td>
</tr>
<tr>
<td>Ours(↓ 2)</td>
<td><strong>0.97s</strong></td>
<td>91.91</td>
<td>78.49</td>
<td>84.67</td>
</tr>
<tr>
<td>Ours(↓ 2) w/o chairs,desktop</td>
<td>0.97</td>
<td>96.95</td>
<td>82.86</td>
<td>89.35</td>
</tr>
</tbody>
</table>

a lab to a kitchen. We ignore one scene - desk2 - as the camera pose obtained gave unsatisfactory results. We compare our results in 2D with [5] in Fig. 3.13. We make an underlying assumption that our precise 3D object proposals can be correctly identified as the relevant objects inside those bounding boxes. As the authors perform object detection, we manually measure the precision and recall for each object individually by using the groundtruth labeling per point as discussed previously. Our performance is significantly better than [5]. We also observed that our technique picks up other objects such as laptop, computer mouse, and kitchen devices as objects of interest which are treated as background in their analysis.

**Our Dataset:** We collected our own dataset using Microsoft Kinect v1. We captured six video sequences varying from 400 frames to 800 frames. Some of the scenes are shown in Fig. 3.14. We placed multiple objects such as a kettle, books, coffee mugs and a messenger bag in various locations in an indoor room. We purposely placed these objects close to each other to showcase that our approach can easily cluster these nearby objects separately. We also placed coffee mugs on the whiteboard attached to the wall (scene 3) to highlight that our approach is also able to recognize them as objects of interest despite not being placed on the table.

Our approach is able to consistently identify small objects such as a power bank or books lying on the table. The plane removal step assists in segmenting out the books placed flat on the table. This would be extremely difficult to do in the global point cloud where the table 3D points may not lie on a unit plane due to depth noise and SLAM errors. However, due to the presence of depth noise, some table pixels are not filtered out and become a part of the nearby objects, which leads to clustering two neighboring objects together as seen in scene 3 in Fig. 3.14 where the power bank and books are clustered together as one object.
3.7 Summary

In this chapter, we have developed a novel multi-view based 3D object proposal technique using depth information along with initial 2D proposals. To our knowledge, this is the first technique that truly produces 3D object proposals without using trained object segmentation and recognition classifiers. There are several ways in which our system can be extended. 3D object proposals can improve the 3D object detection, and can improve camera pose using these objects as known entities in the scene. It can also assist AI in figuring out areas of importance in the scene without human input.

In future work, we aim to optimize our system towards true real-time 3D object proposals over even larger environments by exploring multi-scale representations for memory and computational efficiency. Ultimately, we intend to integrate our
Figure 3.14: Left column shows the point cloud of various scenes in our dataset. 2\textsuperscript{nd} column represents our top ranked filtered points and 3\textsuperscript{rd} column shows the resulting 3D object proposals based on density based filtering. Our approach can locate and cluster even thin objects such as books. However, due to flying-pixel depth noise, sometimes two extremely close objects are clustered together.

system with SLAM to improve its accuracy by treating the object proposals as fixed landmarks in the scene.
CHAPTER 4

SPIRAL STITCHING OF TUNNEL IMAGES

Traditional image stitching algorithms use transforms such as homography to combine different views of a scene. They only work when the scene is planar or when the camera’s orientation is varied while keeping its location static. This severely limits their use in real world scenarios where an unmanned aerial vehicle (UAV) flies around in the scene. In this chapter, we exploit Structure-from-Motion (SfM) along with bundle adjustment (BA) and known scene geometry to robustly create cylindrical images captured in a given environment such as a cylindrical tunnel where the camera moves forward in a spiral fashion. The captured images of the inner surface of the given scene are combined to create a composite panoramic image that is textured onto a 3D cylinder in Unity to create an immersive environment for users to navigate through and inspect specific parts of the tunnel.

4.1 Introduction

The rapid development of low-powered Unmanned Aerial Vehicles (UAVs) and drones has introduced a need for automatic stitching of the captured images in a given environment. Such applications not only help in visualizing the scene but may also help in other areas such as creating Dense Surface Models (DSM) [62] and fault detection [63] that may be present.

Aging infrastructure is becoming an increasing concern in the developed countries. There is a need for automatic or user-assisted assessment, diagnosis and fault detection of old structures such as sewage tunnels, bridges and roof-tops. Some of these structures may also be inaccessible or too dangerous for human inspection. For example, manual inspection of deep tunnel networks is an extremely challenging and risky task due to the inaccessibility and potentially hazardous environment contained in these tunnels. Due to the health risks involved, UAVs
provide a perfect choice in such cases as they are compact and can be automated or controlled by a user to remotely capture the necessary information.

This chapter focuses on imaging and inspection of the Deep Tunnel Sewerage System (DTSS). DTSS is a massive integrated project currently being developed by the Public Utilities Board (PUB) in Singapore to meet the country’s long-term clean water needs through the collection, treatment, reclamation and disposal of used water from industries, homes and businesses [6]. These DTSS tunnels are covered with a corrosion protection lining (CPL) for protection. The aim of this chapter is to stitch the images collected by the UAV into a cylindrical panoramic view of the tunnel and render the tunnel in 3D to inspect the physical conditions of the CPL as well as the structural integrity of the tunnel as a whole.

While UAVs provide a viable alternative for remote assessment of deep tunnels as they are unaffected by debris and sewage flow, they are primarily designed for high altitude aerial imagery and are not appropriate for short range detailed imaging of tunnel surfaces. One alternative is to attach a 360° camera in front of a UAV and capture the panoramic view of the tunnel. However, these 360° images have low resolution that are not suitable for fault detection. Moreover, most of these cameras are too heavy and/or consist of odd shapes that render them difficult to attach to a UAV. Instead, we use a lightweight and high resolution GoPro HERO4 camera. The camera rotates around the shaft of the UAV while the UAV moves forward in a tunnel, in turn providing us with spiral-like images. We provide an angular resolution of 0.0325° per pixel and approximately 1.70mm per pixel resolution for lateral movement in a tunnel of radius 3m.

This chapter presents a framework where we use a UAV to fly through a cylindrical sewage tunnel as shown in Fig. 4.1. We integrate depth information captured by depth sensors secured with the UAV along with 2D color images captured by a rotating camera moving forward in a spiral fashion as shown in Fig. 4.2. We utilize Structure-from-Motion (SfM) along with bundle adjustment (BA) and exploit scene geometry to automatically stitch the captured images into cylindrical images. Thereafter, the stitched cylindrical images are textured on a tunnel-like 3D object and displayed in unity [64] to assist users in visualization, remote inspection, and fault detection of these tunnels.

In particular, we make the following contributions:

• A novel 360° revolving camera system is proposed which, coupled with the maneuverability of the UAV, allows capturing high definition images
of the tunnel surface efficiently in a unique spiraling motion. Advanced lightweight time-of-flight infrared sensors are integrated in the UAV to provide all-around proximity detection and localization.

- We use a custom wide-baseline feature matcher to estimate camera pose between neighboring images. During the BA process, we exploit known scene geometry to robustly improve the initial estimated camera poses. These images, with estimated camera pose, are then used to perform seamless stitching to obtain a cylindrical image of the given scene.

- A geometrical visualization framework is developed using Unity to assist the users in visualizing the tunnels and fault detection.

We evaluate our approach on both a synthetic dataset where ground-truth information is available, and real data obtained using a rotating camera setup. In both cases, we demonstrate that our proposed framework is capable of handling real-life situations where sensors may contain noise and the UAV may hover around in the tunnel.

This chapter is structured as follows. In Section 4.2, we review other works that are related to our research topic. In Section 4.3, we provide a brief overview of cylindrical projection and how it is related to our problem. In Section 4.4, we discuss how we use depth sensors to locate the UAV in a cylindrical tunnel and estimate a maximum speed limit of the UAV so as to obtain a full panoramic view of the tunnel without any holes (skipped regions) in the tunnel. Section 4.5 presents the proposed cylindrical stitching and refinement steps in detail, highlighting our contributions and observations at each stage. We report our results using both simulated data and real data in Section 4.6. Finally, in Section 4.7 we conclude this chapter and discuss future research directions.

4.2 Related Work

**Image Stitching:** A lot of work has been done in the computer vision and photogrammetry community [65–68] to perform image stitching. Traditional image stitching techniques rely on an underlying transform, usually a $3 \times 3$ affine or homographic matrix that maps pixels from one coordinate frame to another. Typical image stitching techniques such as AutoStitch [69] assume the camera’s location...
Figure 4.1: (a) Deep tunnel sewers [6]. (b) View from inside the tunnel [6]. (c) Our custom UAV design. (d) The UAV descends into the tunnel from one access shaft, flies through the tunnel and ascends to the surface from the next access shaft.

...to be static, i.e. a pure camera rotation between captured images, or the captured scene to be roughly planar. In our panoramic spiral imaging system, the camera both rotates and translates while capturing the scene. This translation is often not negligible as compared to the distance of the tunnel surface to the camera. Moreover, the planar assumption of the scene is invalid for us since we capture images of a cylindrical tunnel. Dornaika and Chung [70] proposed a heuristic approach of piecewise planar patches to overcome this issue. Other recent methods [71], [72], [73] propose a different strategy of aligning the images partially in order to find a good seam to stitch different images in the presence of parallax. However, these methods rely heavily on reliable feature detection and matching, which might be difficult in the tunnel environment. Furthermore, these methods do not exploit known geometry of the scene. Hence, we propose to use known scene geometry, robust feature matching and measured camera’s orientation and location for high quality image stitching.
Structure-from-Motion: Structure-from-Motion (SfM) refers to the recovery of 3D structure of the scene from given images. A widely known application of SfM is where ancient Rome is reconstructed using Internet images [74, 75]. SfM has made tremendous progress in the recent years [76–79]. SfM is highly dependent on accurate feature matching to estimate the camera’s pose and location. Since, our camera rotates significantly across consecutive images, we use a wide-baseline feature matcher to estimate the camera’s orientation and location. RANSAC [80] is typically used for outlier detection which unfortunately results in discarding a large fraction of true matches [81]. This results in a fragmented and discontinuous reconstruction [82]. We use a robust wide-baseline feature matcher - RepMatch - that uses an epipolar guided feature matcher to guide the discovery of more feature matches.

Bundle Adjustment: In general Bundle adjustment (BA) refers to the problem of jointly refining and estimating optimal 3D structure and camera(s) intrinsic and extrinsic parameters - camera calibration and pose. The seminal paper [83] showcases its use in computer vision and related fields. Classically, BA is formulated as a non-linear least squares problem [84, 85]. The cost function is assumed to be quadratic in terms of 2D reprojection error. Outlier detection and removal is used to make it more robust to noise and missing data.
While such a formulation makes BA extremely general, it relies heavily on a good initial solution for convergence to global minimum. Although recent works [86] have tried to make it faster, it is still a considerably slow optimization process. We leverage the known geometry to make our system more robust and faster. During the triangulation process, we aggressively discard points with a large 2D projection error. We also detect points that are farther than the expected geometry by a certain threshold and remove them as well. This not only makes the BA results more accurate, it also results in a faster convergence as there are considerably fewer 3D points to fit in the non-linear least squares problem.

### 4.3 Cylindrical Projection

In this section, we set up the cylindrical projection that models spiral imaging of tunnel surfaces. As shown in Fig. 4.3, a generic 2D pixel of an acquired image, \([u, v]^{\top}\), can be projected to a 3D point \(X = [x, y, z]^{\top}\) using a camera’s intrinsic projection parameters - focal length, \(f\), and optical center \(([c_x, c_y]^{\top})\) - as follows:

\[
X = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \lambda K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} \frac{u-c_x}{f} \\ \frac{v-c_y}{f} \\ 1 \end{bmatrix} \tag{4.1}
\]
where $\mathbf{K}$ represents the internal calibration matrix of the camera and $\lambda$ refers to the pixel’s depth. This 3D point is projected onto a unit cylinder as follows:

$$
\begin{bmatrix}
\hat{x} \\
\hat{y} \\
\hat{z}
\end{bmatrix} = \frac{1}{\sqrt{X^2 + Z^2}} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}
$$

(4.2)

A 3D point lying on a unit cylinder is represented by two parameters - angle, $\theta$ and height, $h$, as follows:

$$
\begin{bmatrix}
\sin \theta \\
h \\
\cos \theta
\end{bmatrix} = \begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix}
$$

(4.3)

The unit cylinder can be unwrapped onto a planar image as shown in Fig. 4.3 as follows:

$$
\begin{bmatrix}
\hat{u} \\
\hat{v}
\end{bmatrix} = \begin{bmatrix} f\theta + c_x \\ fh + c_y \end{bmatrix}
$$

(4.4)

An example of cylindrical projection of a 2D planar image captured by a camera is shown in Fig. 4.4.
4.4 Our Framework

In this section, we first provide a brief motivation of our UAV design. We then discuss how we move the UAV to the center of the tunnel initially before the data capturing process starts. We also provide a mathematical limit to how fast the UAV can move so as to obtain a full panoramic view of the tunnel without any holes (skipped regions) in the tunnel.

4.4.1 Deep-Tunnel Imaging System

We use the evolution of capsule endoscopy for advanced bio-medical imaging as major motivation of designing our UAV. Traditionally, the camera is placed along the longitudinal axis of the capsule to capture the front view. Because of the direction in which the camera is mounted, they are normally paired with wide angle lens to maximize the field of view. However, building a UAV for our purpose comes with a few restrictions. As seen in Fig. 4.1(c) the UAV is designed to carry additional sensors - depth and gas sensors. The depth sensors are used for obstacle avoidance and ensuring the UAV flies in the center of the tunnel. The gas sensors are placed in front to detect the presence of harmful toxic gases that can be present in the tunnel. Moreover the fixed pitched propellers placed in the front and back of the UAV along with safety padding underneath these propellers create an obstructed view for a front or back facing camera. Moreover, the images captured by the 360° cameras similar to the ones used in capsule endoscopy suffer from low resolution and serious distortion due to the wide angle lens. As we want high fidelity images for detecting aberrations in the tunnel linings, such cameras are unsuitable for our requirement.

Our approach alleviates this problem by re-positioning the camera to a location where there is no obstructed view and image distortion is minimal - perpendicular to the intestine surface (UAV shaft). However, this results in a reduction of the field of view (FoV). We overcome this problem by rotating and capturing the image data while the UAV flies forward as shown in Fig. 4.2. Using this approach, the entire inner cylindrical surface of the tunnel can be imaged with high fidelity and minimal optical distortion using a single rotating camera. A cylindrical 2D panoramic view of the tunnel can be reconstructed from the overlapping images captured by the rotating camera moving forward in a spiral fashion.
4.4.2 Sectional Location of the UAV in the Tunnel

When the UAV is lowered into the shaft, the first task is to estimate the camera’s initial location. We use the depth measured by the range sensors mounted on the UAV. These range sensors measure the distance of the UAV vertically and horizontally as shown in Fig. 4.5 and have a range up to 14 m. We use the built-in inertial measurement unit (IMU) to stabilize the UAV such that the UAV is pointing horizontally forwards, i.e. zero pitch, yaw, and roll. Since the tunnel might contain flowing water or sludge at the base, we discard the readings provided by the depth sensor pointing below and use the depth information from remaining three sensors. We analyze the cross-section of the tunnel as shown in Fig. 4.5.

Assume the 3D location of the UAV to be $p$. Let the distance measured by two horizontal and a vertical depth sensor be $\{d_1, d_2, d_3\}$ respectively and the radius of tunnel be $r$. We present a framework to compute the offset of the UAV from tunnel center, $[t_x, t_z]^T$. We parameterize the three points located by the three depth
sensors as follows:

\[
p_1 = \begin{bmatrix} r \sin \alpha \\ h \\ r \cos \alpha \end{bmatrix};
\]

(4.5)

\[
p_2 = \begin{bmatrix} r \sin(\alpha + \theta) \\ h \\ r \cos(\alpha + \theta) \end{bmatrix};
\]

(4.6)

\[
p_3 = \begin{bmatrix} r \sin(\alpha + \beta) \\ h \\ r \cos(\alpha + \beta) \end{bmatrix};
\]

(4.7)

Since the UAV is stabilized with zero yaw, pitch and roll, we can assume \( h = 0 \) without loss of generality. As the UAV lies on \( (p_3 - p_2) \), we can parameterize the 3D point \( p \) as follows:

\[
p = \begin{bmatrix} t_x \\ 0 \\ t_z \end{bmatrix} = \gamma p_1 + (1 - \gamma)p_2
\]

(4.8)

\[
= \begin{bmatrix} r(\gamma \cdot \sin \alpha + (1 - \gamma) \cdot \sin(\alpha + \theta)) \\ 0 \\ r(\gamma \cdot \cos \alpha + (1 - \gamma) \cdot \cos(\alpha + \theta)) \end{bmatrix}; \gamma \in [0, 1]
\]

where \( \{\theta, \alpha, \beta, \gamma\} \) are four unknown variables. A generic circle can be defined by three given points. However, in our case, since we know the radius of the tunnel, we need only two points to locate the UAV’s position in the tunnel. We use the depth sensor information and the radius of the tunnel as follows:

\[
||p - p_1||_2 = d_1;
\]

(4.9)

\[
||p - p_2||_2 = d_2;
\]

(4.10)

\[
\cos \theta = \frac{r^2 + r^2 - (d_1 + d_2)^2}{2 \cdot r \cdot r} ;
\]

(4.11)

\[
90^\circ - \frac{\theta}{2} = \alpha - 90^\circ;
\]

(4.12)

\[
||p - p_3||_2 = d_3;
\]

(4.13)

\[
(p - p_3) \times (p_1 + p_2) = 0
\]

(4.14)
This results in an over-constrained solution for \( \{\theta, \alpha, \beta, \gamma\} \). We use ordinary least squares (OLS) [87] to estimate the parameters. Additionally, since our system is over-constrained, we can continue to compute the location of the UAV in case one range sensor fails during the data capturing process. This saves precious time as sending the UAV inside a tunnel is extremely taxing in terms of time and money.

4.4.3 Maximum Speed of UAV

Assume the UAV’s camera to be positioned at a distance \( r_1 \) from the center of the tunnel. The camera moves horizontally in \( y \) direction. Assume the camera only moves forward without any rotation. The camera captures an image of resolution \(-[nr \times nc]\). We intend to estimate the maximum allowable distance, \( d \), such that the two images of a cylindrical tunnel with radius, \( r \), contain some overlap so they can be stitched together without having any dead space (black holes) between the two images. As shown in Fig. 4.6, the distance between the first and last pixels (in \( y \) direction) of an image is the highest allowable distance moved by the camera. We can extrapolate the 3D points represented by the pixel locations - \([1, 1]^\top\) and \([1, nr]^\top\) - as follows:

\[
\begin{bmatrix}
  1 \\
  1
\end{bmatrix} =
\begin{bmatrix}
  f \frac{X_1}{Z_1} + c_x \\
  f \frac{Y_1}{Z_1} + c_y
\end{bmatrix};
\begin{bmatrix}
  1 \\
  nr
\end{bmatrix} =
\begin{bmatrix}
  f \frac{X_{nr}}{Z_{nr}} + c_x \\
  f \frac{Y_{nr}}{Z_{nr}} + c_y
\end{bmatrix}
\]

where \([X_1, Y_1, Z_1]^\top\) and \([X_{nr}, Y_{nr}, Z_{nr}]^\top\) represent the 3D points located on pixels \(-[1, 1]^\top\) and \([1, nr]^\top\) respectively, \( f \) represents the focal length of the camera and \([c_x, c_y]\) represents the optical center of the camera. Since we want at least some overlap between two consecutive images the maximum distance that the UAV can move is represented by:

\[
Y_1 = \frac{(1 - c_y)Z_1}{f}
\]

\[
Y_{nr} = \frac{(nr - c_y)Z_{nr}}{f}
\]

\[
d_{max} = \Delta Y = Y_{nr} - Y_1 = \frac{(nr - 1)Z}{f}
\]
where

\[ Z_1 = Z_{nr} = Z' = r \cos \theta - r_1 \]

Using the law of sines as seen in Fig. 4.6(a):

\[ \frac{r}{\sin(\pi - \frac{\Omega_h}{2})} = \frac{r_1}{\sin(\frac{\Omega_h}{2} - \theta)} \]

\[ \implies \theta = \frac{\Omega_h}{2} - \arcsin\left(\frac{r_1}{r} \sin\left(\frac{\Omega_h}{2}\right)\right) \]

(4.17)

where \( \Omega_h \) refers to the horizontal field of view of the camera. The maximum horizontal movement allowed is:

\[ d_{\text{max}} = \frac{(nr - 1)(r \cos \theta - r_1)}{f} \]

(4.18)

\[ d_{\text{max}} \approx 2 \tan\left(\frac{\Omega_v}{2}\right)(r \cos \left(\frac{\Omega_h}{2} - \arcsin\left(\frac{r_1}{r} \sin\left(\frac{\Omega_h}{2}\right)\right) - r_1\right) \]

(4.19)

where \( \Omega_v \) refers to the vertical field of view of the camera. Given \( n \) images per full 360° rotation of the camera, the maximum movement allowed per image is represented by:

\[ \dot{d}_{\text{max}} \approx \frac{2 \cdot (r \cos \theta - r_1) \tan(\frac{\Omega_v}{2})}{n} \]

(4.20)

4.5 Our Approach

We utilize a GoPro HERO 4 camera mounted on UAV for the image stitching process. The rotating camera mounted on the UAV’s shaft captures images after rotating a certain degree every few milliseconds. The benefits of taking an image after the camera rotates a certain pre-defined rotation is to avoid issues such as motion blur and rolling shutter noise.

As we use state-of-the-art IMU and optical flow sensors for UAV navigation the UAV moves forward in a roughly straight line throughout the data capturing
Figure 4.6: (a) The UAV is allowed to move a maximum distance of $r_1$ mm from the center of the tunnel. (b) As this distance from center of tunnel increases, the maximum allowable distance moved by the UAV decreases to allow some overlap in between two images with the same camera rotation and ensure no dead-space during the offline stitching process.

process. We can also estimate the distance moved by the UAV per image using the IMU. Using accurate motors and actuators in our rotating mechanism enables us to rotate the camera with an accuracy of a degree or less. We consider this rotation measurement and translation of the camera as an initial camera pose for our framework.

4.5.1 Camera Pose Estimation - A Case for RepMatch

Camera pose includes the change in rotation and translation between two adjacent frames captured by a moving camera. Usually this camera pose is estimated using Structure-from-Motion (SfM) techniques. An image is processed to detect and identify local features such as SIFT [88] or SURF [89]. The best candidate match for each feature point is found in the adjacent image. Usually, all matches in which the distance ratio, called ratio test, is greater than 0.8 are rejected as false matches. Thereafter, RANSAC is used to identify and reject outliers. Camera pose is estimated using the remaining inliers.

SfM has made tremendous progress recently [74, 75]. Unfortunately, current SfM systems struggle in accurate pose estimation and 3D reconstruction of modern architecture that consists of repeated structures such as floor tiles and brick walls. Thus, we use RepMatch [7] to estimate camera pose as it is shown to perform better in presence of repetitive man-made structures. It uses an epipolar
Figure 4.7: (a) SURF matching points after outlier rejection when camera rotates 36°. (b) RepMatch matching points when camera rotates 36°. (c) SURF matching points after outlier rejection when camera moves forward. (d) RepMatch matching points after outlier rejection when camera moves forward. SURF based matching struggles in presence of repeated structures such as floor tiles while RepMatch gives us more dense accurate point matching which results in a more reliable camera pose.

guided matcher which postpones selection of the correct pose at a later stage. This allows RepMatch to reliably validate the very large but noisy set of all matches which contains many previously discarded true matches. Essentially, RepMatch couples Bilateral Functions (BF) [81] and RANSAC outlier rejection schemes by relaxing the ratio test to get lots of initial matching points and using RANSAC coupled with epipolar geometry to identify locally correct matches to refine matching points. We show a comparison of standard SfM using SURF feature point matching and RepMatch in Fig. 4.7.

While RepMatch provides us with geometrically correct dense matching, it makes a baseline assumption that no pixel in the scene is stationary with respect to the camera. In our case, this can prove fatal as we use a camera placed on a moving trolley to collect our dataset. This trolley is partially visible in some images captured by the camera. As the trolley is rigidly connected to the camera, it is stationary with respect to the camera. This can lead to wrong matches as shown in Fig. 4.8. We overcome this problem by using a manual mask to avoid areas where we do not expect point matches as we have a rough estimation of the camera’s rotation and translation.
Figure 4.8: We show a few examples of RepMatch [7] point matching. In (a) and (b) RepMatch correctly identifies matching point giving us a good camera pose estimate. However, in scenes represented by (c) where objects such as the trolley carrying the camera move along with the camera, RepMatch incorrectly identifies points lying on these as matching points resulting in a noisy camera pose. We avoid these wrong matches by using a manual mask.

4.5.2 Bundle Adjustment

Bundle Adjustment (BA) involves triangulating the matching points to estimate their 3D location and minimizing the 2D projection error across all images containing these matching points.

Three-step Pruning: Occasionally RepMatch provides us with erroneous matching points if there are moving objects in the scene as shown in Fig. 4.8. We perform a three-step pruning before feeding the matching points to BA. First, we use a manual mask to identify regions where we do not expect any matching points. As we know the approximate rotation and translation between frames, we reject any matching points in these regions. Second, we reject any matching points that have a reprojection error of greater than three pixels. Third, we use known geometry of the scene to identify badly triangulated points. A 3D triangulated point is rejected if it is at a distance greater than a predefined threshold from the expected geometry of the scene. This three-step pruning results in a significantly lower average reprojection error as seen in Table 4.1.

Linking Rotational and Horizontal Movements: So far we estimate the pose between two consecutive frames where the camera goes through a significant rotation as shown in Fig. 4.9(a). We observe that if the camera moves below a certain
Table 4.1: The reprojection error reduces significantly after perform three-step pruning on baseline RepMatch.

<table>
<thead>
<tr>
<th></th>
<th>Horizontal movement</th>
<th>Rotational movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline RepMatch</td>
<td>6.07</td>
<td>5.85</td>
</tr>
<tr>
<td>After three-step pruning</td>
<td>0.83</td>
<td>0.63</td>
</tr>
</tbody>
</table>

Figure 4.9: (a) Matching between rotationally consecutive frames. (b) Matching between both rotationally consecutive frames and horizontally moving forward frames. We add links between horizontally moving forward frames (when camera completes a full 360° rotation). This allows for pose estimation between these additional frames making the bundle adjustment step more robust.

4.5.3 Estimating Translation Scale

The camera pose obtained using any standard SfM technique is devoid of actual scale of the scene. Usually the translation component of the camera pose is a unit vector and needs to be scaled to obtain the correct camera pose. While using multiple cameras in a system helps estimating this scale, this remains a challenging problem to estimate in monocular camera setup. We utilize the scene geometry
Introducing the vertical links between every $i$ and $i + 10$ frames significantly lowers the 2D reprojection and 3D error during the Bundle Adjustment process.

To estimate the scale per image pair. The color camera is rigidly attached with a depth sensor that provides us a scalar depth measurement per image. We use this depth information to estimate the scale of the camera’s translation per image pair. We select a reliable pair match between two consecutive (rotational or horizontally moving forward) images that lie in the center of the image. This ensures that the depth measurement being used from the depth sensor is indeed accurate for this pair match:

$$X_1 = \lambda_1 \cdot K^{-1} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}; \quad ||X_1||_2 = d_1$$

$$X_2 = \lambda_2 \cdot K^{-1} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}; \quad ||X_2||_2 = d_2$$

$$RX_1 + \hat{\lambda}t = X_2$$

where $K$ refers to the intrinsic calibration matrix of the camera; $[R, \hat{t}]$ represent the estimates camera pose; $[x_1, y_1]^\top$ and $[x_2, y_2]^\top$ are pair matches with $d_1$ and $d_2$ being their corresponding depth measurements. This provides us with a reliable estimate of the translation of camera between two consecutive images.
4.5.4 Cylindrical Image Stitching

Let us first assume that the UAV is stationary and only the rotating camera rotates to capture images in the tunnel. The image capturing and stitching design is shown in Fig. 4.5.

Using the camera pose estimated in the previous step enables us to project and transform the 2D pixel information per image into world coordinate frame. Let $X_w$ represent the 3D points, per frame, in world coordinate frame. A global $i^{th}$ point $X^i_w$ lying on the cylindrical tube of radius $r$ can be represented by two parameters - $\theta^i$ and $h^i$ as follows:

$$X^i_w = \begin{bmatrix} r \sin \theta^i \\ h^i \\ r \cos \theta^i \end{bmatrix}$$  \hspace{1cm} (4.21)

Let $X_c$ represent the 3D points in camera frame of reference for every image captured by the rotating camera. Every $i^{th}$ pixel ($=[u^i, v^i]^T$) can be projected onto 3D as:

$$X^i_c = \frac{z^i_c}{f} \begin{bmatrix} u^i - c_x \\ v^i - c_y \\ f \end{bmatrix}$$  \hspace{1cm} (4.22)

where $z^i_c$ refers to the depth of $i^{th}$ pixel which is unknown. $X_c$ and $X_w$ are related by:

$$X_w = RX_c + t$$  \hspace{1cm} (4.23)

where $R$ is a $3 \times 3$ orthonormal rotation matrix and $t = [t_x, t_y, t_z]^T$ represents the translation of the UAV in world coordinate frame. Initially the UAV is assumed to be at the center of the tunnel ($t = \vec{0}$). As the UAV moves horizontally across the tunnel, $t_y$ increases. $t_x$ and $t_z$ denote the deviation of the UAV from center of the tunnel in $\vec{x}$ and $\vec{z}$ directions respectively.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \hspace{1cm} t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

79
For every \(i^{th}\) pixel, we obtain three equations for three parameters - \(\{z_c^i, \theta^i, h^i\}\). The two equations involving \(\theta\) can be deterministically solved as follows:

\[
\sin \theta = (RX_c)(1) + t_x
\]

\[
= \frac{z_c}{f}(r_{11}(u - c_x) + r_{12}(v - c_y) + r_{13}f) + t_x
\]

\[
\cos \theta = (RX_c)(3) + t_z
\]

\[
= \frac{z_c}{f}(r_{31}(u - c_x) + r_{32}(v - c_y) + r_{33}f) + t_z
\]

where \((RX_c)(1)\) and \((RX_c)(3)\) represent the first and third row of matrix \(RX_c\) and:

\[
z_c = f(r \cos \theta - t_z)
\]

\[
= \frac{f(r \cos \theta - t_z)}{(r_{31}(u - c_x) + r_{32}(v - c_y) + r_{33}f)}
\]

Thus,

\[
r \sin \theta = \frac{(r \cos \theta - t_z)(r_{11}(u - c_x) + r_{12}(v - c_y) + r_{13}f) + t_x}{(r_{31}(u - c_x) + r_{32}(v - c_y) + r_{33}f)}
\]

Let

\[
a = \frac{r(r_{31}(u - c_x) + r_{32}(v - c_y) + r_{33}f)}{(r_{11}(u - c_x) + r_{12}(v - c_y) + r_{13}f + t_x)}
\]

\[
\sin \theta = \alpha \implies \cos \theta = \pm \sqrt{1 - \alpha^2}
\]

Hence, we can simplify Eq. 4.26:

\[
a \sin \theta - r \cos \theta + t_z = 0
\]

\[
a \alpha - r(\pm \sqrt{1 - \alpha^2}) + t_z = 0
\]

\[
a \alpha + t_z = r(\pm \sqrt{1 - \alpha^2})
\]

\[
a^2 \alpha^2 + t_z^2 + 2at_z \alpha = r^2 - r^2 \alpha^2
\]

Resulting in a quadratic equation:

\[
(a^2 + r^2)\alpha^2 + 2at_z \alpha + (t_z^2 - r^2) = 0
\]

Solving for this quadratic equation gives us two solutions for \(\alpha\). This results in
four possible solutions for \( \theta \) \((\sin \theta = \alpha = \sin(\pi - \theta))\). We use the orientation of the rotating camera for the current frame to select the correct solution from the four possible solutions. Thereafter, we can also estimate the \( y \) component of \( \mathbf{X}_c \) by:

\[
h^i = \frac{z_i^j}{f} \left( r_{21}(u - c_x) + r_{22}(v - c_y) + r_{23}f \right) + t_y
\] (4.30)

Once we compute the world coordinate of every pixel’s location, we can obtain the cylindrical projection for it as described in Sec. 4.3.

4.6 Experimental Results

In this section, we perform synthetic and real experiments and compare our results in both noiseless and noisy scenarios. We also discuss our Unity framework and display a few examples of the rendered cylindrical scene in Unity for visualization.

4.6.1 Synthetic Data Results

We used blender [90] to render a hollow cylinder and imported it into Unity [64] to generate synthetic data. The cylinder of radius, \( r = 3 \) m, is positioned such that the geometrical center of the cylinder is located at \([0, 0, 0]^\top\). We texture-mapped a brick wall and a panoramic view of Seattle’s skyline onto the inner face of the cylinder for visualization purposes. The light source is fixed to look vertically down for our experimental evaluation. Hence, images captured around 0\(^{\circ}\) rotation are brightly lit while the images captured around 180\(^{\circ}\) appear dark due to lack of illumination. A few images captured by the virtual camera are shown in Fig. 4.11. Let us denote the cylindrical image to be synthesized as \( \mathbf{X}_{2D}^w \).

**Stationary Camera Positioned at Center:** In our first experiment, we positioned the camera at the center of a cylindrical tunnel. The camera is held stationary throughout this experiment and only rotated rotated by \( \beta = 30^{\circ} \) per frame and images are captured consequently. An example of this process is shown in Fig. 4.11. It takes 12 images to complete the full 360\(^{\circ}\) rotation. Thereafter, the images are stitched together as described in Sec. 4.4. In this experiment:
\[
R = \begin{bmatrix}
\cos(\beta k) & 0 & \sin(\beta k) \\
0 & 1 & 0 \\
-\sin(\beta k) & 0 & \cos(\beta k)
\end{bmatrix}; \ t = \begin{bmatrix}
0 \\
0 \\
0
\end{bmatrix} \quad \forall k = \{0, 1, \ldots, 11\}
\]

(4.31)

For each image captured, we can use Eqs. 4.29 and 4.30 to project each pixel onto the cylindrical stitched image \(X_{w}^{2D}\). However, performing this “forward warping” may leave some holes in the stitched image. Thus, instead of performing forward warping, we use the four corners of each image to obtain the forward warped boundary. Thereafter, for every pixel inside this boundary of \(X_{w}^{2D}\), we perform “inverse warping” to obtain its pixel location and intensity information in \(X_{c}^{i}\). The fully stitched image is shown in Fig. 4.12(e). We observe that the images align perfectly with each other and all the “curved” bricks are straightened after performing cylindrical projection.

**Stationary Camera Positioned off Center**: Under ideal conditions, the camera should be positioned at the center of cylindrical tunnel. However, in reality this will not happen as UAVs tend to hover around and might have difficulty maneuvering to the center of the cylinder before each image is captured. The position of the camera with respect to the center of the tunnel can be measured...
Figure 4.12: Top row displays the images captured from unity when the camera is rotated from 0 to 360 degrees. Bottom row displays the stitched image obtained after our stitching process.

using depth sensors mounted on the sides of the UAV or by using SfM and tracking the UAV’s movement per frame. The camera is held stationary throughout this experiment and only rotated by $\beta = 30^\circ$ per frame and images are captured consequently. We move the camera off center to $[0.5m, 0m, 0.5m]^T$. In this experiment:

$$R = \begin{bmatrix} \cos(\beta k) & 0 & \sin(\beta k) \\ 0 & 1 & 0 \\ -\sin(\beta k) & 0 & \cos(\beta k) \end{bmatrix} ; \quad t = \begin{bmatrix} 0.5 \\ 0 \\ 0.5 \end{bmatrix} \quad \forall k = \{0, 1, \ldots, 11\} \quad (4.32)$$

A cylindrical image is a $360^\circ$ view of the scene around the camera flattened on a planar image. Thus, even though the camera is off center, we can still view what the cylindrical projection of the scene looks like assuming the UAV’s current position to be the center of the tunnel as shown in Fig. 4.13(a). While the stitching is perfect, the straight lines (bricks) are no longer straight and we can see the zoom in and zoom out effect when the camera is far or near to the tunnel boundary respectively. We use Eqs. 4.29 and 4.30 to synthesize what each image would look like if the camera was positioned at the center of the tunnel. The fully stitched image is shown in Fig. 4.13(b). We observe that the images align perfectly with each other and all the “curved” bricks are straightened after performing cylindrical projection.

**Varying the Speed of UAV:** In this experiment, we vary the speed of the UAV to empirically show that Eq. 4.20 provides us with a reasonable bound for maximum UAV speed for a full panoramic stitch of the tunnel without any holes. We used a camera of focal length 3800 pixels, in a tunnel of radius 3 m. The virtual camera’s
Figure 4.13: (a) Stitched cylindrical image without accounting for camera location off center. (b) Stitched cylindrical image after accounting for camera’s location off center.

Figure 4.14: Stitched cylindrical image when the camera moves forward by (a) 18 cm per rotation. (b) 20 cm per rotation. We only stitch the 0° and 180° images to highlight the gaps between views. As the maximum allowable speed of the camera is 18 cm per rotation, we start observing holes in the cylindrical image if the camera’s speed increases above the maximum allowable speed.

FoV is \([\Omega_h, \Omega_v] = [60^\circ, 101^\circ]\). Using Eq. 4.20, we obtain a maximum allowable speed of approximately 18 cm per rotation for the UAV. We simulated the camera to move horizontally forward at 18 cm per rotation and 20 cm per rotation. We stitch the 0° and 180° images only to showcase the impact of UAV’s speed on the stitching process in Fig. 4.14. We observe that when the camera moves 18 cm per rotation, we obtain a perfect stitch without any gaps between the images. However, as soon as we increase the speed to 20 cm per rotation, we start seeing holes between the projected images as the camera is moving too fast and skips certain sections of the tunnel.
Testing Robustness of Camera Pose estimation: In our last synthetic experiment, we aim to simulate UAV movements in real-world conditions. A UAV is expected to suffer from jitters and sideways movements while it tries to balance itself and move forward in the tunnel. This means that the camera’s movement and rotation per image given by IMU may be unreliable for our stitching purposes.

We simulate a tunnel of radius 3 m. We initially positioned the camera at the center of the cylindrical tunnel. Our baseline movement and rotation in the three orthogonal directions are \( t = [0, 0.15, 0]^t \) and \( r = [0, 0.524, 0]^t \) respectively. This means that we expect the ideal movements in the tunnel to be 15 cm horizontally forward (y direction) with a 30° rotation across y axis. We add Gaussian noise with zero mean and standard deviation of [5, 2, 5]\(^t\) to our translation and Gaussian noise with zero mean and standard deviation of 2° to our rotation per image. We run the simulation till the camera completes ten rotations and record the groundtruth translation and rotation of the camera per frame. We perform two stitches - baseline and using wide baseline camera pose estimation using RepMatch [7].

In the baseline stitch, we assume that the camera does not suffer from any jitter or sideways movement and blindly trust the initially planned rotation and translation per frame. In our second approach, we use RepMatch to perform wide-baseline feature point matching across two consecutive images and estimate the camera pose using these matching points. The results of the groundtruth, baseline and RepMatch stitch are shown in Fig. 4.15. The baseline stitch, understandably, fails to provide us with a good stitch of the Seattle panoramic view. However, using RepMatch wide-baseline matching improves the camera pose significantly and we get an extremely good stitch. The stitch obtained using RepMatch is slightly different than the groundtruth on the left side as seen in Fig. 4.15(c). This cylindrical projection stitch can be wrapped around a cylinder in the Unity engine to provide an immersive 3D display of the panoramic view for users.

4.6.2 Real Data Results

We use a GoPro HERO4 camera to capture our dataset. We used this camera because it is extremely lightweight and can be used seamlessly with a depth sensor to simultaneously capture an image and scalar depth information and save these measurements on an external storage device. As this GoPro camera suffers from
Figure 4.15: We add random jitter and movement to the camera after each image capture. Using baseline rotation and translation leads to an extremely inaccurate and incoherent stitch. Using SfM and BA enables us to improve the camera pose per image pair and results in a much more accurate and coherent cylindrical projection of the scene.

Heavy distortion, we first used a checkerboard to obtain its intrinsic calibration parameters including the distortion parameters. The images are pre-processed to remove the distortion before the stitching process in our setup.

We developed a rotating mechanism to rotate the internal shaft of the UAV by approximately $36^\circ$ per rotation. This results in ten images per full $360^\circ$ rotation. We attached the GoPro HERO4 camera and a depth sensor to the UAV. This system provides us with a color image along with a depth reading per image. Currently, the rotating mechanism is built on a tripod stand until we have a functional UAV.

We used an underpass to capture our dataset as a “proof of concept”. The underpass consists of a horizontal floor, vertical walls on both left and right sides and a cylindrical ceiling. We measured the cross-section of the underpass i.e. the width of the floor, vertical height of the walls and height of the ceiling. This provides us with known geometry that is used during the stitching process. An example of the underpass is shown in Fig. 4.16(a).

We perform two experiments using our setup. In the first experiment, we rotate the camera to capture a panoramic $360^\circ$ view of the scene before it is manually moved horizontally forward providing us with a “cylindrical” data capture. This step is repeated ten times. In the second experiment, we move the tripod stand
manually after each image to obtain a “spiral” data capture. This resembles the actual data capturing process we expect to achieve when we have an automated flying UAV. In both these experiments, we carefully measure the forward movement of the camera after each image. We present our results and observations as follows:

**Cylindrical Data Capture**: In our first experiment, we placed the tripod consisting our camera setup in the underpass. The camera is initially positioned to look vertically downwards. The camera is then rotated $90^\circ$ to capture four images and four depth measurements. This provides us with an initial position of the camera inside the underpass.

Afterwards, the camera is allowed to rotate approximately $36^\circ$ per image and captures ten such images to complete a full rotation of the underpass from its current location. Thereafter, the tripod is moved forward manually. We repeat this
process for seventeen such rotations. During this process, the tripod, consisting of the camera setup, moves roughly 5 m in the underpass. We manually measure the forward horizontal movement of the system after each full 360° rotation of the camera. This provides us with a reasonable estimate of the camera’s rotation (36°) and the camera’s translation between the two consecutive images.

We stitch the images by using two different approaches. In the first approach, we utilize the known geometry of the scene to identify if a pixel in an image lies on the floor, right, left, or ceiling of the underpass. Thereafter, these four regions are stitched separately. This process leads to obtaining four images - floor, right wall, ceiling, and left wall. These images are then textured onto four planes in unity to obtain a 3D representation of the scene.

In the second approach, we repeat the process of identifying which part of the underpass each pixel belongs to. Thereafter, it is projected onto a unit cylinder as we discussed in previous sections. This results in a panoramic cylindrical view of the scene. The cyclic panoramic image is then wrapped around an internally hollow cylinder in unity to display the panoramic view in unity. The cylindrical projection and 2D stitch of the wall murals is shown in Fig. 4.17.

**Spiral Data Capture:** In our second experiment, we repeat the initial localization process to obtain the camera’s initial position in the underpass.

Afterwards, the camera is rotated 36° and moved forward manually. This results in the “spiral” data capturing process that we obtained in our simulated dataset. We repeat this process for fifty times obtaining five full 360° rotations of the camera. During this process, the tripod, consisting of the camera setup, moves roughly 5 m in the underpass. We manually measure the forward horizontal movement of the system after each image is captured by the camera. This provides us with a reasonable estimate of the camera’s rotation (36°) and the camera’s translation between two consecutive images.

We use [81] to estimate the camera pose between two consecutive frames using the strategy discussed above. We also obtain the camera pose between every tenth frame (1 and 11, 2 and 12 etc.) as discussed in Sec. 4.5.2. Thereafter, the camera pose for each image along with matching points are used in a bundle adjuster. We used the MATLAB bundle adjuster to perform this step. This provides us with a more accurate camera pose per frame. Figure 4.18 displays the 3D reconstruction of matching floor points with and without these steps to improve camera pose. We observe that connecting every tenth frame and removing wrongly matched points is extremely critical for a good pose estimation. We also stitch the images by
Figure 4.17: We perform a stitch of the underpass using RepMatch [7]. The camera rotates full 360° before it is manually moved forward. The top row displays the cylindrical stitch of the underpass and the scene rendered in a rectangular room in Unity.

using two different approaches we have discussed in the previous section. The 2D stitch of the wall murals in shown in Fig. 4.19.
Figure 4.18: (a) 3D reconstruction of floor points after 60 frames without removal of wrongly matched points. (b) 3D reconstruction of floor points after 120 frames after removal of wrongly matched points. (c) 3D reconstruction of floor points after 230 frames after removal of wrongly matched points. The floor points still roughly lie on a plane after the camera moves 22 m forward demonstrating the accuracy and robustness of our SfM and BA process.

Figure 4.19: We perform a stitch of the underpass using RepMatch [7]. The camera rotates 36° and manually moved forward per frame. We display the left and right walls after cylindrical stitching.

4.6.3 Visualization in Unity

We display the cylindrical images of the scene rendered as described in previous sections in Unity. Unity is a game engine used mainly to create platform inde-
pendent video game applications. However, it also provides us with a good set of tools to provide an immersive 3D visualization for our purposes. We envision our system as a “fly-through” of the scene. The user controls the camera position using keyboard and mouse and just like a first-player shooter game, can float around in the 3D scene freely. This enables the user to not only view the rendered tunnel images but also move closer to the areas where the user suspects faults in the tunnel. Fig. 4.20 shows our setup for straight and curved tunnels. The user provides a text file with the curve of tunnel and the cylinders for the curved tunnel are rendered when the user starts the application. We also render the cylindrical projection of our real data results. We show the front and left and the right views of the tunnel after zooming out of the tunnel for a better visualization of the scene in 3D in Fig. 4.21.

4.7 Conclusion

We presented a simple and accurate system to capture images of a given scene using a spirally moving camera system and display the panoramic stitched images in unity for an interactive 3D display. The presented method excels in scenes where prior geometrical information is available. This allows us to project the images in 3D and warp them onto a unit cylinder to obtain unit cylindrical images of the scene.

There are various places where our system can be improved. The bundle adjuster often fails to improve the camera pose if the initial camera pose is too far off from the actual camera pose. The bundle adjustment process is also extremely slow for a long stitch. Moreover, we rely on the prior knowledge of the scene geometry. In future, we plan to improve our bundle adjustment process to improve the camera pose and use additional depth sensors mounted on the sides of UAV to sense the 3D geometry of the scene during the data capturing process.
Figure 4.20: A hollow cylinder is created in Blender and UV mapped. This cylinder is imported in unity to create both straight and curved tunnels based on the user requirements. The user can move freely inside the tunnel and move closer to the boundaries to carefully inspect for any damage in the tunnel.
Figure 4.21: The “Cylindrical” dataset is rendered in Unity. We add front and end walls to signify start and end of the tunnel passage.
CHAPTER 5

SUMMARY AND CONCLUSION

In this thesis, we focused on depth cameras and their applications. In particular, we looked at three main problems.

In Chapter 2, we discussed the noise properties of depth cameras and the standard calibration scheme. We proposed a novel algorithm that is simple yet accurate to simultaneously denoise depth data and self-calibrate depth cameras. The presented method excels in estimating calibration parameters when only a handful of corners and calibration images are available, where the traditional approach really struggles.

In Chapter 3, we developed a novel multi-view based 3D object proposal technique using depth information along with 2D object proposals as input. To our knowledge, this is the first technique that truly produces 3D object proposals without using trained object segmentation and recognition classifiers. We performed a detailed analysis of each step of our algorithm using standard metrics such as detection rate, average precision and recall. We showed that every step is critical to obtain precise online 3D object proposals.

In Chapter 4, we exploit Structure-from-Motion (SfM) along with bundle adjustment (BA) and known scene geometry to robustly stitch images captured in a cylindrical tunnel where the camera moves forward rotating in a spiral fashion. The captured images of the inner surface of the tunnel are combined to create a composite panoramic image that is textured onto a 3D cylinder in Unity for users to navigate through and inspect specific parts of the tunnel.

Ultimately, we believe that the ceiling for depth based applications is sky high and the scientific community is just getting started. Currently, the community is using depth sensors as an external measurement for improving computer vision techniques. We believe that ultimately, depth sensors will work in tandem with various integrated sensors to create immersive 3D applications.

The future for research in depth related applications is extremely bright.
APPENDIX A

WORKINGS OF A TIME-OF-FLIGHT CAMERA

We provide a brief overview of time-of-flight (ToF) technology that is used by the Kinect v2 and PMD camcube.

A.1 Time of Flight

Time-of-flight techniques measure the time taken for a signal to travel from the source, strike an object and traverse back to the receiver. Light detection and ranging (LIDAR) and radar are based on this principle. Once we know the time taken by the signal, we can compute the distance, $d$, as follows:

$$d = \frac{s \cdot t}{2}$$  \hspace{1cm} (A.1)

where $s$ and $t$ are the speed of the signal and time taken by the signal to reach the receiver respectively. As the signal needs to cover twice the distance (to and from), the actual depth of the scene is half of the distance travelled by the signal.

To achieve millimeter accuracy, the time-of-flight needs to be recorded with picosecond accuracy, which is only possible if we use hardware that is shielded and kept cool. Even then, such hardware is prone to noise. Thus, the current ToF based depth cameras (PMD camcube) measure depth using a slightly different concept. PMD camcube modulates the signal and measures the phase difference between the received and sent signal. The camera uses a state-of-the-art four-phase-shift algorithm [91], [92] to compute the depth values at each pixel. The signal is sampled four times per period at equal intervals and these sample points are summed over thousands of periods, considerably increasing the accuracy and signal-to-noise ratio of the depth measurement at each pixel. One drawback of the modulation technique is that it limits the range of the camera. If an object is sufficiently far away from the receiver, the phase can warp around zero before
being measured. This leads to a lower estimation of depth of the scene compared to its true value. The modulation frequency used by PMD camcube is 20 MHz, resulting in a maximum range of 7.5 m. In the following subsection, we briefly describe the four-phase-shift algorithm and motivate the type of noise present in the ToF camera’s depth measurement.

A.1.1 ToF Depth Camera - PMD Camcube

This state-of-art technology is known as a photon mixing device (PMD). Rather than using a laser beam to estimate one depth value for the object, the entire scene is illuminated with modulated NIR. Due to the device’s similarity to CCD cameras, these grid-measurements are often referred to as “smart pixels” [8]. Due to current limitations of hardware, these pixels are roughly 10 times larger than standard CCD pixels. This severely limits the size of the ToF depth cameras. The current size of the PMD camcube 2.0 is 204 × 204 pixels only. We used this camera to generate real depth data and perform calibration for this thesis.

Figure A.1 shows an illustration of a “smart pixel” present in PMD camcube. It is a five-terminal device with two light-sensitive photogates in the middle. The electrons move to the left or right diodes based on the difference in output voltages between the two channels. This process is modeled as a correlation between the sent modulation signal and the received signal. The sent modulation signal is assumed to be a rectangular wave, and the received signal, being a low-pass IR-LED signal, is assumed to be a sinusoid.

\[
\begin{align*}
    s(t) &= \sum_{n=-\infty}^{\infty} rect \left( \frac{t}{T} - 2n \right) \\
    r(t - T_L) &= a_0 \cos(wt - wT_L) + B
\end{align*}
\]

Here, B refers to the received average incident light (background light and dc component of the light source), \(a_0\) refers to the amplitude of modulated light, and \(T_L\) is time taken by the infrared rays to go “to and fro” from the object and strike the receiver. The cross correlation function is given by:
\[
\phi(\tau) = s(t) \otimes r(t)
\]
\[
= \frac{k}{T} \int_{t=-\frac{T}{2}}^{T/2} s(t)r(t+\tau)dt
\]
\[
= k \left[ \frac{a_0}{\pi} \cos(\omega T_L) + \frac{B}{2} \right]
\]

where \( k \) refers to the number of periods per integration time. The function is evaluated at four different phases \( \omega T_L \), namely \( 0^\circ, 90^\circ, 180^\circ, 270^\circ \). This allows us to compute a unique solution for the phase.

\[
\psi = \arctan \left( \frac{\phi(270^\circ) - \phi(90^\circ)}{\phi(0^\circ) - \phi(180^\circ)} \right)
\]

We can also compute two other important values per pixel: signal strength, \( a_0 \), and estimated intensity, \( B \):

\[
a_0 = \pi \sqrt{\frac{(\phi(270^\circ) - \phi(90^\circ))^2 + (\phi(0^\circ) - \phi(180^\circ))^2}{2}}
\]

\[
B = \frac{\phi(0^\circ) + \phi(90^\circ) + \phi(180^\circ) + \phi(270^\circ)}{4}
\]

Figure A.2 provides us a good visual insight to the physical meaning of these parameters. The amplitude \( a_0 \) signifies the strength of the signal at that pixel. Thus the greater the amplitude, the more reliable the depth measurement. Hence, it can be used as a threshold to discard unreliable or noisy depth readings, which is potentially extremely useful in applications such as ICP [93], [58]. The background
Figure A.2: The cross correlation function for a single modulation period for PMD camera [8].

Illumination $B$ provides us the estimated intensity information at each pixel which can be used for calibration and denoising. The depth per pixel can be computed using the phase and prior knowledge of the modulation frequency used.

$$d = \frac{\psi}{2\pi} \cdot \frac{c}{2f_{mod}}$$
REFERENCES


