

Environment Reconstruction from an Aerial Perspective with RGB-D and Fisheye Cameras

Ritika Shrivastava



Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-154

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-154.html>

May 20, 2022

Copyright © 2022, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

**Environment Reconstruction from an Aerial Perspective with RGB-D
and Fisheye Cameras**

by Ritika Shrivastava

Research Project

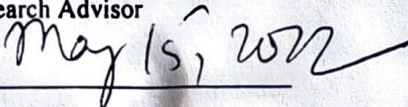
Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for the
degree of **Master of Science, Plan II.**

Approval for the Report and Comprehensive Examination:

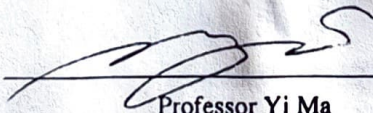
Committee:



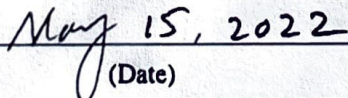
Professor Shankar Sastry
Research Advisor



(Date)



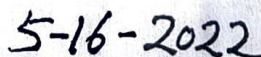
Professor Yi Ma
Second Reader



(Date)



Professor Allen Yang
Second Reader



(Date)

Environment Reconstruction from an Aerial Perspective with RGB-D and Fisheye Cameras

by

Ritika Shrivastava

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Masters of Electrical Engineering and Computer Science
in
Electrical Engineering and Computer Science
in the
Graduate Division
of the
University of California, Berkeley

Committee in charge:

Professor Sastry, Chair
Professor Yang

Spring 2022

Abstract

Environment Reconstruction from an Aerial Perspective with RGB-D and Fisheye Cameras

by

Ritika Shrivastava

Masters of Electrical Engineering and Computer Science in Electrical Engineering and
Computer Science

University of California, Berkeley

Professor Sastry, Chair

Drones are becoming increasingly prevalent due to their affordability, agility, and size. With this increased usage it is important to account navigability in a variety of terrains. This required detailed understanding of the environment. Most 3D environment reconstruction techniques use LiDAR or RGB-D cameras. However, LiDAR is too expensive and heavy for drones and RGB-D cameras have limited a field of view. To improve upon this we worked on a hardware design with a fisheye camera and RGB-D camera for 3D environment construction. This paper also explores improvements to SLAM module results through trajectory alignment. The use of least-squares estimations of transformation parameters given two points is shown it improve the rotational and translation error.

To my parents, my brother, and grandparents for always believing in me

Contents

Contents	ii
List of Figures	iii
List of Tables	iv
1 Introduction	1
2 Related Works	4
3 Hardware Setup	6
3.1 Problem Setup	6
3.2 Designing the Mount	7
3.3 Calibration	9
3.4 Frame Synchronization	13
3.5 Conclusion	14
4 Software Setup	15
4.1 System Design	15
4.2 Results	18
4.3 Future Improvement	19
5 SLAM Experiments	21
5.1 SLAM Experiments - Trajectory Alignment	21
6 Conclusion	24
Bibliography	25

List of Figures

1.1	Pinhole Camera model	2
1.2	Fisheye Camera model	2
1.3	Camera System view 1	3
1.4	Camera System view 2	3
1.5	Camera System view 3	3
3.1	View from fisheye Camera	6
3.2	View from ZED2 Camera	6
3.3	Aerial view of mount 1	8
3.4	Side view of mount 1	8
3.5	Labeled Aerial view of mount 1	8
3.6	Fisheye Camera's FOV is obscured by ZED. The grey region shows the region from the fisheye camera's FOV that is covered by the back of the ZED.	8
3.7	Aerial view of mount 2	9
3.8	Side view of mount 2	9
3.9	Labeled Aerial view of mount 1	9
3.10	Aerial view of mount 3	10
3.11	Side view of mount 3	10
3.12	Labeled Aerial view of mount 3	10
3.13	Aerial view of mount 4	11
3.14	Side view of mount 4	11
3.15	Labeled Aerial view of mount 4	11
3.16	Distorted fisheye camera image	13
3.17	Undistorted fisheye camera image	13
4.1	Software System Overview	15
4.2	Scene 1 image	18
4.3	Point cloud generated by the ZED2 for Scene 1	18
4.4	Scene 2 image	18
4.5	Point cloud generated by the ZED2 for Scene 2	18
4.6	Room 2 Reconstruction	19
4.7	Room 2 Reconstruction	19

List of Tables

3.1	Fisheye camera calibration flags and corresponding errors	12
5.1	Root-mean-squared error in translation and rotation on Vicon room sequences from the Euroc MAV dataset. The bold results show the result with the smallest error. The data with TA has trajectory alignment applied to it.	22

Acknowledgments

This paper was written thanks to the guidance and inspiration of my advisors Professor Sastry and Professor Yang. I would like to thank them sincerely for all of their continual support and valuable insights.

Motivation to do this masters came from my family. My Nanu (Ram Mohan Prasad) always wanted a engineer in the family, and was very proud when I was admitted to Berkeley. My Dadi (Shail Sagar Srivastava) always blessed me to continue to learning and be knowledgeable. This work would was possible their blessings alongside my Baba (Prem Sagar Srivastava) all bestowed from heaven. I would also like to thank my Nani (Archana Prasad) for her love and affection.

My mother and father always motivated me to strive to do my best. Their dedication, perseverance and sacrifices have brought me this far. My sincerest thanks and gratitude goes to them for standing by me and supporting me. My brother's endless stream of memes and Kpop tunes have given me a much needed laugh and relaxation in times of need. Thank you Mumma, Daddy, and Devam for your love and affection.

This work would also not be possible without the help of Amay Saxena and Chih-Yuan (Frank) Chiu whose 8 am meetings on Wednesday help motivate me to learn more about SLAM and structure from motion. Additional thanks go out to Jasmine Bae and Sarthak Madan who were part of the ISSACs group and worked alongside me on the drone project. We faced several highs and lows, yet together we stuck through.

Thank you Stella Seo, Josephine Koe, Kyna Ha, Cathy Chao, Aishwarya Kaimal, and Karan Jain for being there to support me throughout this journey. Your treasure chest of snacks, gossip, and affection have brought me to this point.

Last, but not least Friday. You may not be able to speak, but your actions travel miles. Thank you for always being my silent supporter. I will give you several walks this summer.

Chapter 1

Introduction

There is a boom in drone usage today due to their use cases in remote sensing, surveillance, film, disaster relief, and delivery. There are several applications for drones in outdoor and indoor environments. This is attributed to their affordability, size, and agility. During aerial navigation, drones are expected to be robust and accurately navigate in variable environments which makes safety become an important question and requires knowledge of the environment such as the locations of navigable spaces. This coupled with the ability for drones to reach and view objects from several locations that are hard to reach for humans makes them useful for recreating an objects and environments that might not have been easy to recreate otherwise. Environment reconstructions with drones have a multitude of use cases which include and are not limited to autonomous navigation and obstacle detection.

There are several sensors that assist with environment reconstruction. Some are active range sensors, which includes LiDAR and radars, that are commonly used by industry for obstacle avoidance. However, these tools can often be fail size and weight constrains needed to be placed on small aerial robot. Additionally, the current cost of these sensors ranges on the higher end. Other sensors that are useful for environment reconstruction include cameras. These adopt a vision-based approach and are optimal for drones due to their size, weight, and cost. The development of cheap RGB-D cameras in the consumer market has made it is possible to retrieve the 3D of the scene without extra computational cost. However, these cameras often have narrow field of view (FoV) which can limit the safety of motions in all directions. Fisheye cameras and other omnidirectional cameras can improve on this limitation by providing a larger FoV. This allows for understanding of a larger scene to recreate a scene which leads to fewer frames needed for a scene. However, fisheye cameras require more complex models than conventional camera models due to their extremely wide FoV and the distortion of the images. Figure 1.1 and Figure 1.2 visualize the Pinhole and Fisheye camera model.

Our work combined the use of RGB-D cameras with a fisheye camera to increase the FoV of the system. We build a mount that could be attached to a drone which has the ability to hold 2 ZED2 cameras and a fisheye camera. For the purposed of our work, we planned to use 2 ZED cameras, however due to hardware restrictions we tested out set-up on 1 ZED2

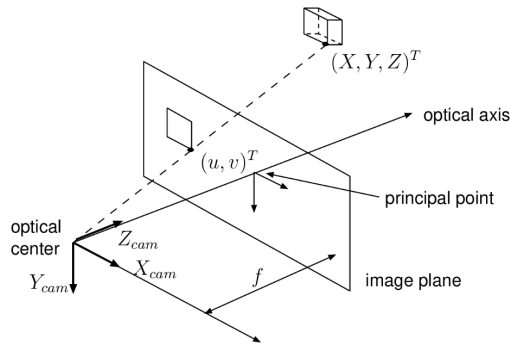


Figure 1.1: Pinhole Camera model

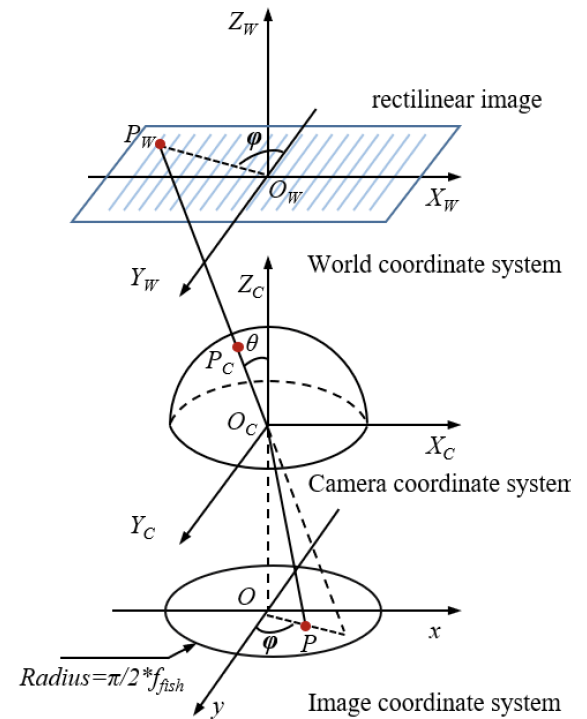


Figure 1.2: Fisheye Camera model

camera and fisheye camera. Despite the difference in FoV for both devices the sections of the images that are shared are valuable. The mount can be seen in Figure 1.3, Figure 1.4, and Figure 1.5.

Additional Work

In addition to the project on Drone-based environment reconstruction with a novel set-up, I also assisted Saxena et al. with their experiments [19]. I have included my contributions here.

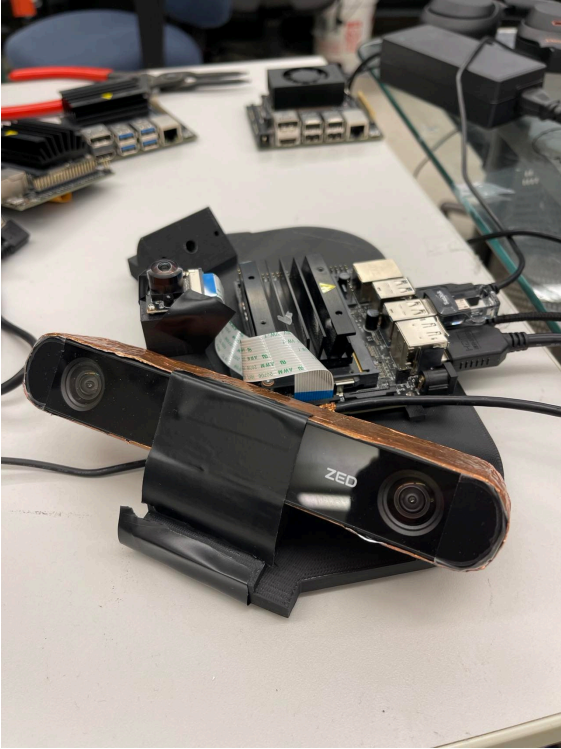


Figure 1.3: Camera System view 1



Figure 1.4: Camera System view 2



Figure 1.5: Camera System view 3

Chapter 2

Related Works

This project has three main areas of work it is related to. The first area is a development of a hybrid camera system with a RGB-D camera and fisheye camera. The second area is structure from motion (SfM) with a fisheye camera on a drone. Lastly, this work is also related to other works which has placed RGB-D cameras on drones for the purpose of environment reconstruction.

Perez-Yus et al. [15] developed a system with a RGB-D camera and a fisheye camera. In their paper, they talked about how their novel setup allows them to take the advantage of the 3D information and scale of the scene obtained from the RGB-D camera and wide field of view to capture a larger area of the scene from the fisheye camera. They stated that this setup has use cases in navigation, SLAM and object detection. This paper explored setting up a combined fisheye and depth camera system and showed that there are benefits to a combined fisheye and depth camera setup.

They expanded on this work with a paper on peripheral depth expansion [16]. This work used the camera setup from the prior work [15] to create an environment reconstruction. From their setup they were able to obtain the depth information for the central field of view of the fisheye camera. Then they identified lines and corners of the room from the fisheye camera. The depth camera was then used to assign probabilities to the existence of the corners detected by the fisheye. Once the corners and lines were validated it was possible to create a recreation of a given room. The results from this paper were good, however they could not be used in all situations. Work by Boutteau et al. also take a similar approach and recreates environments by identifying lines in the fisheye camera's view [3].

While these papers have good results there is one major flaw. These papers mention that their work could only be applied to rooms and scenes which uphold the Manhattan assumption [4]. This assumption states that a given scene is built on a Cartesian grid and is therefore proven to hold true for city and indoor scenes. However, a drone can be flying in several environments (indoor and outdoor) and the Manhattan assumption would be too limiting for this use case. Therefore instead of taking the approach of detecting line in the fisheye to do environment reconstruction, we decided to take the approach of structure from motion on the fisheye cameras.

Another area of related work is using a fisheye camera to perform structure from motion. Gao et al. presented a work with a dual-fisheye setup on a drone with one fisheye camera pointing upwards (towards the sky) and one pointing down (to the ground) with the purpose of autonomous drone navigation and environment reconstruction [6]. They were able to show that with their system could achieve omnidirectional visual perception in real-time. Their program worked in a variety of environments including texture-less environments without prior knowledge. One drawback of this work was that in several scenarios the exposure of the upward-facing and downward-facing did not match, with the upward facing fisheye visual field having a higher intensity of than the downward-facing camera. For stereo matching it is essential that brightness of the visual scenes be similar. For our project, we understood that in a drone (outdoor) scenario most important features are going to be facing downward. For this reason, our approach moved away for a dual-fisheye drone system and focused on a hybrid camera system that would be pointing downward from the drone.

Additionally, Jagannatha et al. explored the usage of the ZED2 camera for environment reconstruction on drones [9]. Their work mounted a ZED2 camera streamed the RGB, Depth, IMU, and clock data over Wifi to a device that developed a 3D reconstruction with the ZED SDK and the OpenARK library. This work showed that it was possible to use a RGB-D, specifically ZED 2, camera on a drone for the purpose of 3D reconstruction.

Chapter 3

Hardware Setup

3.1 Problem Setup

For the purposes of our goals, we decided to have a novel camera setup. Our hybrid camera system was created by rigidly coupling a fisheye camera and 2 ZED2 cameras. This can be seen in Figure 1.3, Figure 1.4, and Figure 1.5. The difference in field of view between the two types of camera is large and can be seen in Figure 3.1 and Figure 3.2. The field of view of the ZED2 camera can be too small for some purposes and applications, especially on drones. To solve this problem, fisheye camera can provide for a larger field of view. There are several papers that discuss the benefits of omnidirectional cameras in the robotics and computer vision space [1] [17]. Additionally, despite their smaller field of view, RGB-D cameras allow for a deeper understanding of the surrounding, with the availability of depth information. This can allow for obstacle detection and environment reconstruction.

In this project, we focused on environment reconstruction on a drone. We tested out two types of fisheye cameras: 160°FoV and 200°FoV and opted for the 200° camera for our work.

The ZED 2 camera is an RGB-D stereo vision camera that is created by StereoLabs. This camera has a dedicated API developed by the company that allows for users to obtain depth maps and optometry data from the cameras. Due to a hardware failure, we were only

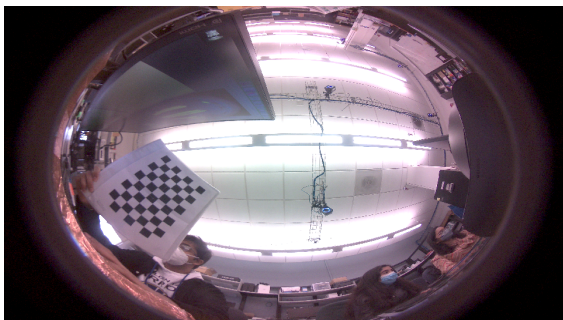


Figure 3.1: View from fisheye Camera

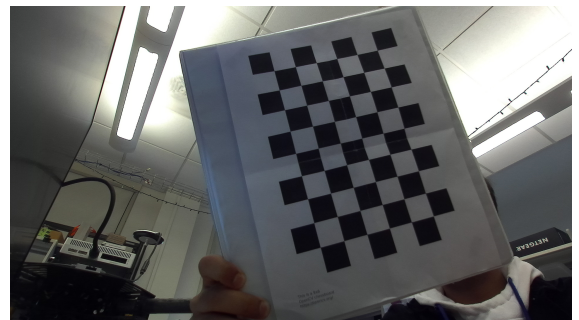


Figure 3.2: View from ZED2 Camera

able to test our implementation on one ZED2 camera. However, this would should be easy to expand to two cameras.

3.2 Designing the Mount

The mount for our project underwent several iterations before reaching the final model. There are a few main goals for the mount. These are: (1) fisheye camera and ZED cameras' field of views should overlap, (2) Jetson Nano should be placed on the mount, and (3) mount should be robust to forces encountered by flight.

The original mount design can be seen in Figure 3.3, 3.4, and 3.5. This design was a flat 3D printed mounts with a few holes for screwing in the ZED and fisheye cameras. This design was a simple print that allowed for mounting of the fisheye and ZED2 cameras. After this mount design it was discovered that the feild of view of the fisheye and the ZED had very little overlap. This was because despite the fisheye having 200° field of view, it was only able to see up to 180°. This was because the mount was blocking the rest. Figure 3.6 shows this. The grey region in this image show the area of the fisheye camera which is blocked by the fisheye camera and the mount. If we continued with this design it would results in a small region of overlap between the view of the fisheye and the ZED.

To solve this problem, we created a second mount design. This mount solved the fisheye field of view problem in two ways: (1) it raised the fisheye camera to be placed higher and (2) angling the ZED. This solves the problem by lifting the fisheye cameras so that the FOV was not blocked by the mount or ZED camera and angling the ZED cameras so that there is a larger overlap in the FOV of the ZED and fisheye cameras. This can be seen in Figure 3.8. The labels locations of the fisheye and ZED camera can be seen in Figure 3.9. While this mount solved the previous problems in the mount, one additional problem faced was that the Jetson Nano was hanging off the edge of the mount. This would not be helpful in a real drone flight. Additionally, the elevated support mount for the fisheye camera broke.

Our third design solved the problems in the second mount by shifting the fisheye camera back, such that the Jetson Nano could be positioned near the center of the mount. We also added supports around the elevated platform for the fisheye camera. This would prevent the fisheye camera's elevated platform from breaking. These changes can be seen in Figure 3.10, Figure 3.11, and Figure 3.12. The performance of this mount was good, There was only one change to be made: the ZED camera needed to be angled more. The issue with this mount was that the region of view for the ZED and fisheye camera has a small region of overlap and the region of overlap on the fisheye has a high rate of distortion.

Our last and final mount design achieved all the desired goals. This mount improved on the previous design by angling the ZED at a 45 degree angle from horizontal. This increased the FOV overlap between the ZED and fisheye camera. The mount can be seen in Figure 3.13, Figure 3.14, and Figure 3.15.

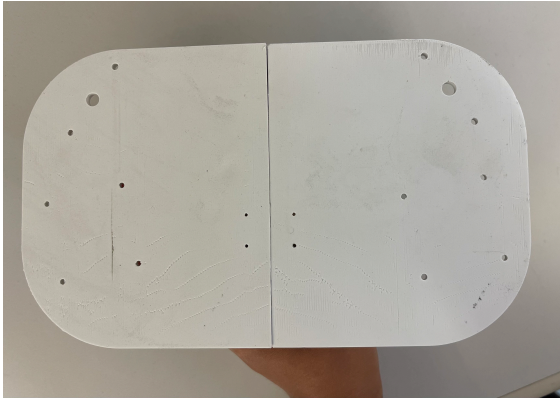


Figure 3.3: Aerial view of mount 1



Figure 3.4: Side view of mount 1

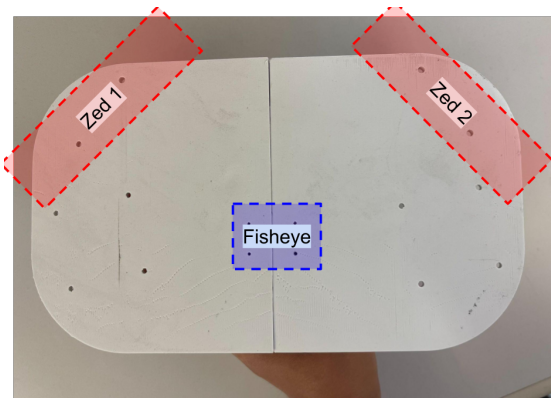


Figure 3.5: Labeled Aerial view of mount 1

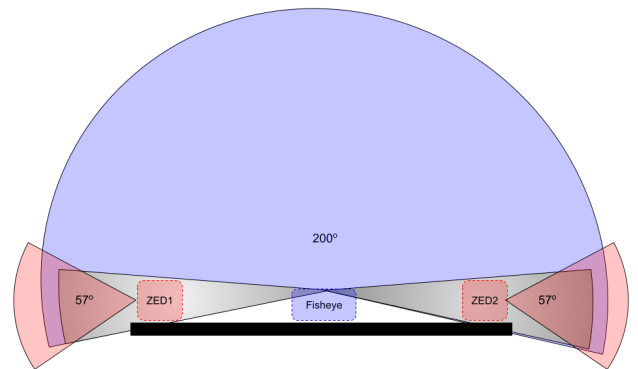


Figure 3.6: Fisheye Camera's FOV is obscured by ZED. The grey region shows the region from the fish-eye camera's FOV that is covered by the back of the ZED.

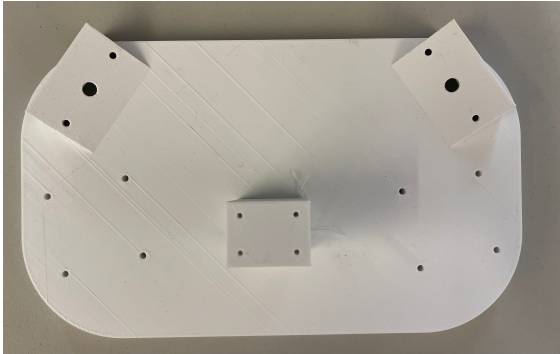


Figure 3.7: Aerial view of mount 2

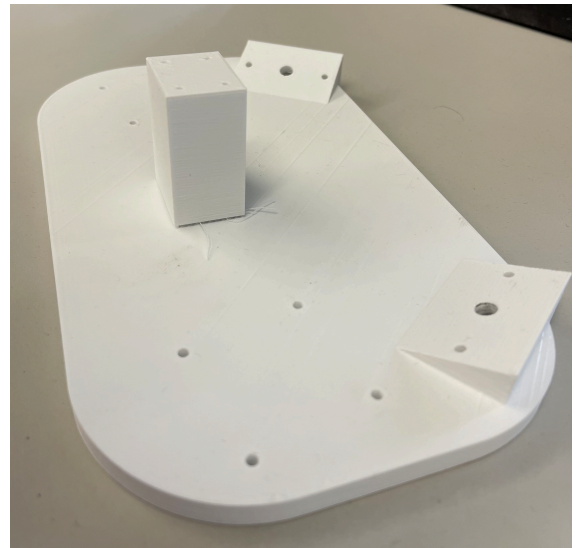


Figure 3.8: Side view of mount 2

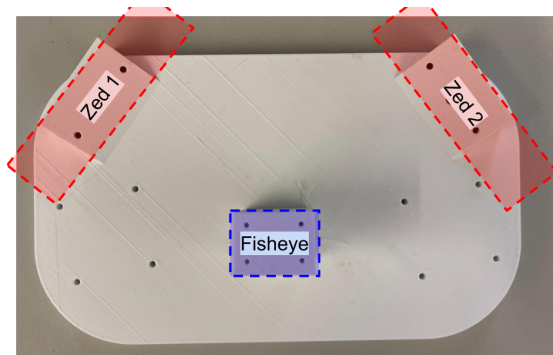


Figure 3.9: Labeled Aerial view of mount 1

3.3 Calibration

ZED2 Camera Calibration

The calibration and distortion matrix for the ZED camera was provided by StereoLabs. This was used in conjunction with functions from OpenCV to calibrate the ZED2 camera.

Fisheye Camera Calibration

For a majority of the project, the IMX219-200 was used as the primary fisheye camera on the mount. This camera has 200° degrees of freedom along the diagonal and has a resolution of 3280 x 2464. The distortion of this camera is less than 11% [8].

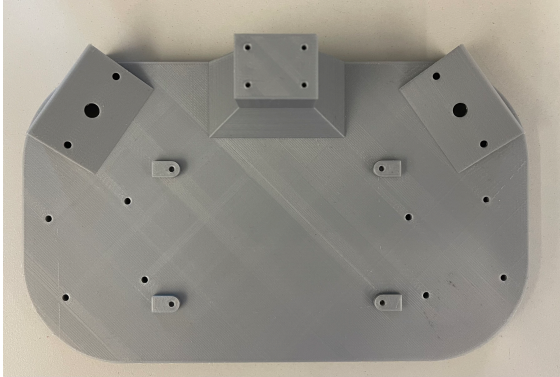


Figure 3.10: Aerial view of mount 3

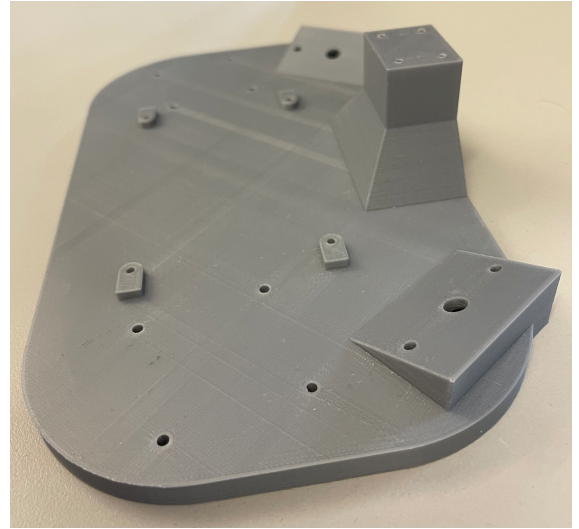


Figure 3.11: Side view of mount 3

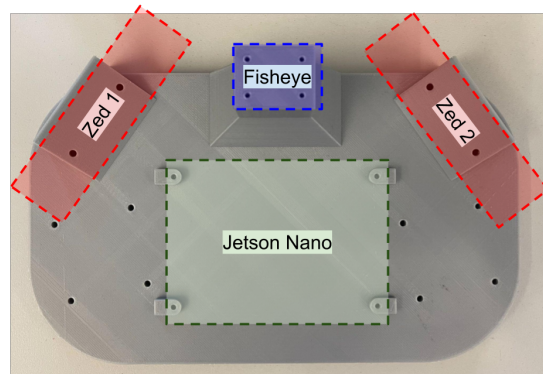


Figure 3.12: Labeled Aerial view of mount 3

When calibrating the camera there were several parameters that needed to be tested. In specific, when using the OpenCV API for calibrating the fisheye, the flag used in the *calibrate* function altered the results of the fisheye calibration [5]. The various possible flags were:

- **CALIB_USE_INTRINSIC_GUESS** : sets the valid initial values for the camera calibration matrix (f_x, f_y, c_x, c_y) .
- **CALIB_RECOMPUTE_EXTRINSIC** : recomputes the extrinsic after each iteration of intrinsic optimization.
- **CALIB_CHECK_COND** : checks the validity of condition number.

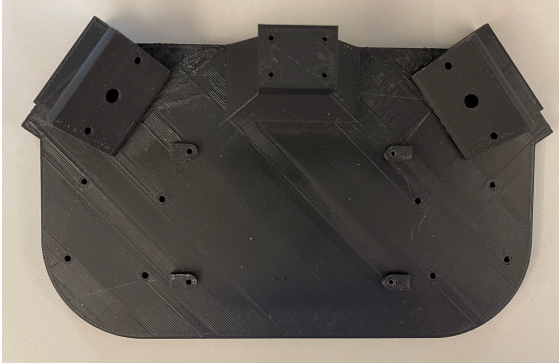


Figure 3.13: Aerial view of mount 4

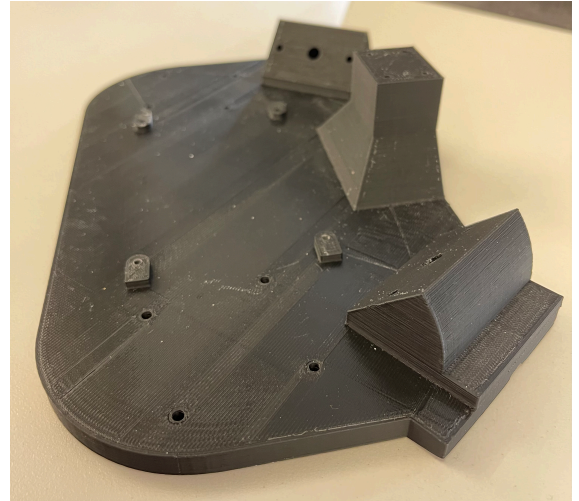


Figure 3.14: Side view of mount 4

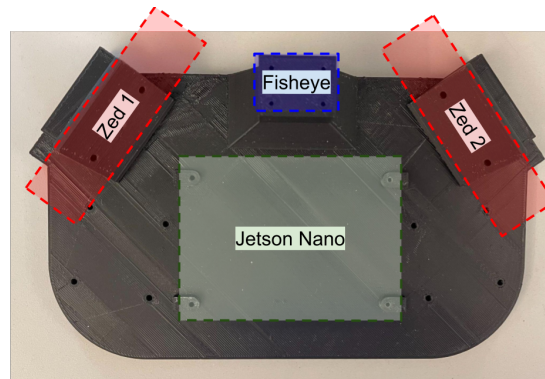


Figure 3.15: Labeled Aerial view of mount 4

- **CALIB_FIX_SKEW** : sets and keeps the skew coefficient (α) at zero
- **CALIB_FIX_K1** : sets and keeps the first distortion coefficient at zero
- **CALIB_FIX_K2** : sets and keeps the second distortion coefficient at zero
- **CALIB_FIX_K3** : sets and keeps the third distortion coefficient at zero
- **CALIB_FIX_K4** : sets and keeps the fourth distortion coefficient at zero
- **CALIB_FIX_PRINCIPAL_POINT** : The principal point is not changed during the global optimization.

To test the performance of the various calibration schemes, we used the average of the sum of the distance between the true location of the point and the projection of that point

Flags	Error
RECOMPUTE_EXTRINSIC + FIX_SKEW	438901.94
RECOMPUTE_EXTRINSIC + FIX_SKEW + FIX_K4	35.77
RECOMPUTE_EXTRINSIC + FIX_SKEW + FIX_K3	7.43
RECOMPUTE_EXTRINSIC + FIX_SKEW + FIX_K3 + FIX_K4	79.91
RECOMPUTE_EXTRINSIC + FIX_SKEW + FIX_K2	4.00
RECOMPUTE_EXTRINSIC + FIX_SKEW + FIX_K1	37.65

Table 3.1: Fisheye camera calibration flags and corresponding errors

using the calculated calibration (K) and (D) matrix. The equation can be written as

$$error = \frac{\sum_{i=0}^n \sqrt{(x_i - \tilde{x}_i)^2}}{n}$$

where n is the number of points in the image used for calibration. x_i is the location of the point in pixel coordinates for the original image. \tilde{x}_i is the calculated location for the new point.

The flags that were tested and resulting the the corresponding errors can be seen in Table 3.1. Calibration was done with points acquired through camera calibration matrix 9x6. From the results above it can be seen that the best flags were RECOMPUTE_EXTRINSIC + FIX_SKEW + FIX_K2. This was what was used for camera calibration. The original fisheye image in grayscale can be seen in Figure 3.16. The results for calibration are seen in Figure 3.17. A noticeable effect of the calibration is that the region of interest in the image has shrunk. However, this is not a problem as long as the region of interest is overlapping with the ZED camera. The FoV can be increased by adjusting the balance in the calibration. That remain as a potential future exploration.

At the end, undistorting the fisheye camera has an error rate for distorting points around the edge of the image. This average difference between points and their undistorted equivalent has an error is around 1.48 pixels in the x axis and 0.45 pixels in the y axis.

Fisheye to ZED2 Calibration

The calibration between fisheye and ZED2 camera calibration involved holding a camera calibration matrix that could be seen in both images, as seen in Figure 3.1 and Figure 3.2. Both cameras were calibrated, then the 8-point algorithm was used calculate the fundamental matrix. We also know that $F = K^{-T}EK^{-1}$, where K is the camera intrinsic matrix and E is the essential matrix. Knowing the essential matrix E allows for the calculation of $R_{zed2fish}$ and $t_{zed2fish}$.

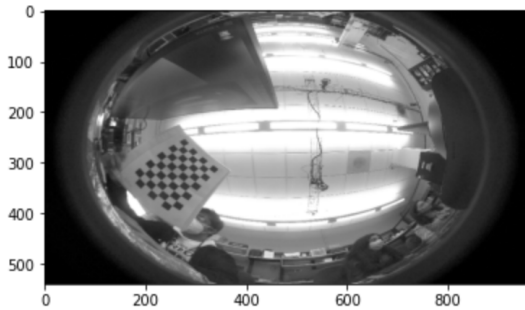


Figure 3.16: Distorted fisheye camera image

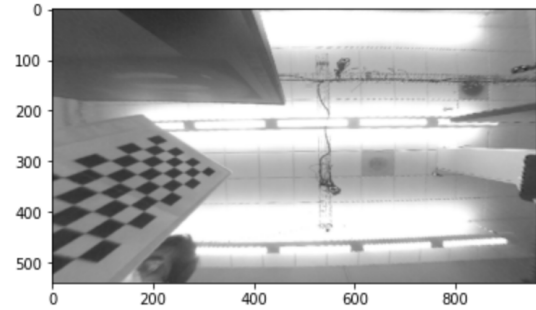


Figure 3.17: Undistorted fisheye camera image

3.4 Frame Synchronization

While working on this project it was important that the frames of the fisheye camera and odometer information from the two ZED cameras were synchronous. This would mean that each R_{zed} and t_{zed} was associated to a frame from the fisheye. To do this, we utilized pynput, time, and python's sub-process libraries. We began by launching a different sub-processes per camera and in our case: 1 python sub-process controlling the fisheye camera, and 2 separate python sub-processes each controlling one of the ZED2 Stereo cameras. These sub-processes launch separate python scripts that handle any camera initialization for there specific cameras and allow us to stream data from all 3 cameras at the same time.

After a sub-process has been started for each camera, we will begin to see the live-streamed data from each camera using each cameras respective libraries(OpenCV or PyZed). Each script is waiting for a cue from the main launcher script before it actually begins to save relevant data. In this case, we used pynput as the cue and waited for the keyboard to click "s". This click is monitored using a pynput keyboard listener by each of the launched scripts as the cue to actually begin saving relevant data. In the case of the ZED 2 cameras this means starting to fused point cloud information, and in the case of the fisheye camera this mean to save image data.

To synchronize reading the point cloud info from the ZED2 and the image data from the fisheye we use the time library to synchronize data capture times between the cameras. Specifically, we allow use the time library to synch a sensor read once per second after the main queue. Once the top of the second is reached, we read our sensor data, then wait until the next second to read the next set of data. Through this synchronization method we are able to ensure that relevant data is fully read within 100ms of each other. We use "s" again to stop the program. Finally, we save all data at the end of the program when the keyboard click presses "q" again using pynput to synchronize the ending of our recording. We save the point cloud information as .ply file and the fisheye image data as a .png file. In addition, we save the relevant R and T of the ZED cameras in a JSON file.

3.5 Conclusion

With the hardware setup mentioned in this section, we were able to begin to develop the software for 3D environment reconstruction.

Chapter 4

Software Setup

4.1 System Design

Our system consisted of 2 parts: (1) data capture and (2) offline 3D reconstruction. Our three sensors capture and store relevant information on the Jetson and we perform 3D reconstruction offline. The overall system can be seen in Figure 4.1.

Data Capture

We used two Stereolabs ZED 2 Stereo RGB-D sensors and a IMX219-200 fisheye camera. For the ZED2 cameras, for every frame we captured the output point cloud and Inertial Measurement Unit (IMU) data. From the fisheye camera, we captured the RGB image at every frame. We collected 960×540 images and point clouds at 1 frame per second which offered a good streaming quality and allowed us to save the point cloud data at each frame.

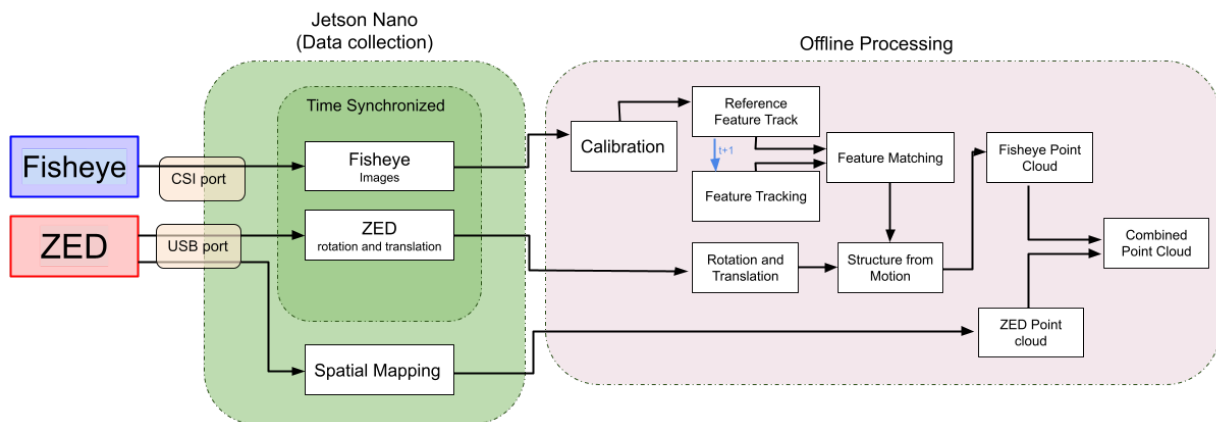


Figure 4.1: Software System Overview

To capture images from similar time frames on the fisheye and two ZED cameras, we used the python tools for multiprocessing and insured that there was a separate pipeline for each camera. Each camera script waits for a cue from main coordinate script before collecting data. Through this pipeline we were able to ensure that frame between cameras occur withing 500 ms.

Once a flight was completed all the data is saved locally. This is transferred offline for post-processing.

3D Environment Reconstruction

The script for 3D Environment Reconstruction can be explained in three components:

1. Structure from motion (SfM) on the fisheye camera
2. Point cloud generation on the ZED
3. Combination of the fisheye-generated point cloud and ZED-generated point cloud

Structure from motion on fisheye camera

To better understand structure from motion (sfm) , we can look to the break down of the concept. The "structure" in can be assumed to be the 3D point cloud of a given scene and the "motion" is the camera location and orientation. The problem of sfm is around getting a point cloud from moving cameras.

For our use case of structure from motion (SfM), we began with a scenario with two image I_0 and I_1 from fisheye cameras, where I_0 is the base image and is where the world reference frame will be set to. This two-view reconstruction with will begin with generating keypoints using the Oriented FAST and Rotated BRIEF (ORB)[18] for both I_0 and I_1 . This feature detector was selected due to its rotation invariance, noise resistance, and speed. These three attributes were essential to optimize for the use case of environment reconstruction on the drone, where rotations a camera's field of view are common. With this we were able to generate features for both images f_0 and f_1 which have the dimensions of $N \times 32$ where N is the number of features generated ($N = 500$ by default) and 32 is number of descriptors used to denote each feature by ORB.

The next step is to match the generated features. To do this the brute force feature matcher is used with the norm hamming flag. This means that when f_0 and f_1 are passed into the the matcher, the output is points pairs which are sorted according to their humming distance in ascending order. This means the $i + 1$ feature pair has a larger humming distance than the i th pair.

Typically after this, the 8-point algorithm can be used to calculate the fundamental matrix. We also know that $F = K^{-T}EK^{-1}$, where K is the camera intrinsic matrix and E is the essential matrix. Knowing the essential matrix E allows for the calculation of the

rotation and translation between I_0 and I_1 , since we know that $E = \hat{t}R$ where \hat{t} is the skew-symmetric matrix of the elements of t .

However, for our purposes, rather than using the approach from the previous paragraph we used the odometer information provided by the ZED2 camera. We can use the rotation and translation provided by the ZED to calculate the transformation between different frames. This can be done with,

$$R_{fish} = R_{zed2fish} \times R_{zed}$$

$$t_{fish} = t_{zed2fish} + (R_{fish} \times t_{zed})$$

where $R_{zed2fish}$ and $t_{zed2fish}$ are known from calibration between the fisheye and ZED camera. This calibration involved holding a camera calibration matrix that could be seen in both images, as seen in Figure 3.1 and Figure 3.2. Then the 8-point algorithm was used calculate the fundamental matrix. We also know that $F = K^{-T}EK^{-1}$, where K is the camera intrinsic matrix and E is the essential matrix. Knowing the essential matrix E allows for the calculation of $R_{zed2fish}$ and $t_{zed2fish}$. Additional information from the equation above include R_{zed} and t_{zed} , which are the rotation and translation obtained by the ZED camera visual odometer.

Once the rotation and translation were calculated, it was possible to triangulate the point in 3D space. This was used to create 3D reconstructions using the fisheye camera.

Point Cloud Generation on the ZED2 Camera

To generate the point cloud we utilized PyZed's Spatial Mapping Parameters as well as the fused point cloud class. Spatial Mapping uses the camera position data to overlap generated point clouds to create one full point cloud. We capture the spatial mapping data once per second using `request_spatial_map_function`. Once we finish capturing the Spatial Map we can call `extract_whole_spatial_map` and then save it as a fused point cloud .ply file. The Fused Point Cloud Class allows us to save this spatial map as a point cloud file.

For setting the Spatial Mapping Parameters we first set the map type to a fused point cloud. We have to set the save texture parameter to true so that we save color data into the fused point cloud. Finally we set the resolution meter parameter to LOW. This allowed us to generate point cloud of objects approximately 5-7 meters away, and fully build the point cloud. We found other resolution meter parameters to be very bad at actually constructing point clouds and were often disconnected. This parameter may depend on the environment you are mapping.

The results can be seen for two different scenes. The first scene is a lab scenario 4.2. The reconstruction for created by the ZED2 camera looks good 4.3. It is easy to make out key features like the yellow cabinet and while it is hard to understand the point cloud from a singular view, this reconstruction was able to recreate the shelves behind the yellow and black cabinets. This point cloud was created by moving the mount through the room as if it was mounted on a drone.



Figure 4.2: Scene 1 image

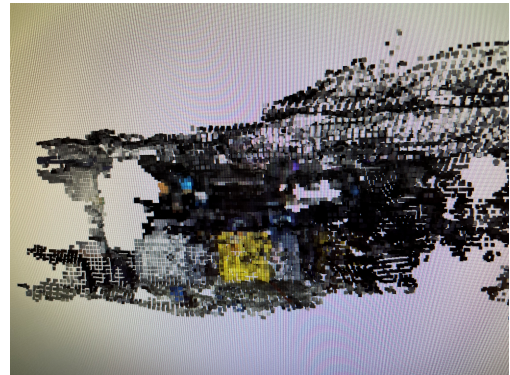


Figure 4.3: Point cloud generated by the ZED2 for Scene 1



Figure 4.4: Scene 2 image



Figure 4.5: Point cloud generated by the ZED2 for Scene 2

The second scene this was done for can be seen in Figure 4.4. The ZED2 point cloud for this can be seen in Figure 4.5. This point cloud is more sparse than the one generated for scene 1. This can be attributed to glass walls are hard to detect as they have no features. Aside from the glass region, the rest of the room has been recreated quite well. There is a brown portion on the left of the image, which is the door. Additionally, some of the pink features of the bookshelf can be seen placed properly on the point cloud. When this point cloud was viewed in 3 dimensions, it was interesting to see the pillar in front of the door also being part of the reconstruction and for it to be placed properly in front of the door.

4.2 Results

For the final results, we worked on environment reconstruction for two different rooms. This is the results from the ZED2 camera. The code from the fisheye camera did not provide a clean point cloud due to problems with time synchronization with the ZED2 camera's

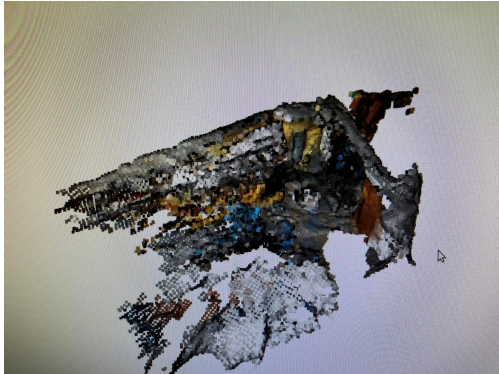


Figure 4.6: Room 2 Reconstruction



Figure 4.7: Room 2 Reconstruction

odometer. The results can be best seen at the following links.

- **Room 1** : [Video recording of the first room](#)
- **Room 1 Reconstruction** : [Video recording of the first room reconstruction](#)
- **Room 2** : [Video recording of the second room](#)
- **Room 2 Reconstruction** : [Video recording of the second room reconstruction](#)

Here the 3D reconstruction can be seen from all sides. In the first reconstruction, we can see that the cabinets can be seen well reconstructed. The yellow cabinet can be seen well reconstructed. In the second room, the bookcase has a good reconstruction, which can be seen from the backside of the bookcase. We can see here that even sections that are outside the room are well reconstructed.

Images of the reconstruction can be seen in Figure 4.6 and Figure 4.7. In the image on the left, the shape of the room is roughly visible. The image on the right shows the prominence of the books on the bookcase. The missing wall was a white wall with very few features. For this reason, it is missing from the reconstruction. If the wall has more features it would also be included.

4.3 Future Improvement

There are a few more steps before this module is complete. The next step will be fix the errors in time synchronization. There are a few extra frames from the fisheye camera which are causing error for the point cloud generation. Additionally, the point clouds from the fisheye camera and ZED2 camera need to be combined.

Additional route for improvement of the project would be fixing the hardware issue with the second fisheye camera and testing the program with both ZED cameras. Increasing the

balance of the fisheye camera is also a test to do for calibrating the fisheye camera that could increased the post-calibration fisheye FoV.

Chapter 5

SLAM Experiments

5.1 SLAM Experiments - Trajectory Alignment

While working on the SLAM module for [19], an additional aspect that was noticed was that the error grew with respect to time. As the time span of the experiment was longer, the error accumulated and caused larger shifts in the data. For this reason, we proceeded to implement an approach for trajectory alignment. This was adapted from the work of Umeyama [20]. This work talks about an algorithm for least-squares estimations of transformation parameters given two points. Given point correspondences in 3-dimensional space which are denoted as $X = \{x_1, x_2, \dots, x_n\}$ and $Y = \{y_1, y_2, \dots, y_n\}$, the minimum value of the root mean squared error can be written as:

$$\sqrt{e^2(R, t, c)} = \sqrt{\frac{1}{n} \sum_{i=1}^n \|y_i - (cRx_i + t)\|^2} \quad (5.1)$$

where R is the rotation, t is the translation, c is the scaling, and n is the number of point correspondences. Umeyama [20] tells that we can calculate estimates for R and t given point correspondences. To reach the calculation for the estimates for R and t we must first define some variables:

$$\mu_x = \frac{1}{n} \sum_{i=1}^n x_i \quad (5.2)$$

$$\mu_y = \frac{1}{n} \sum_{i=1}^n y_i \quad (5.3)$$

$$\sigma_x^2 = \frac{1}{n} \|x_i - \mu_x\|^2 \quad (5.4)$$

$$\sigma_y^2 = \frac{1}{n} \|y_i - \mu_y\|^2 \quad (5.5)$$

Data	MSCKF(n=5)	iMSCKF(n=5)	OKVIS(n=10,k=5)	OKVIS(n=3,k=5)	SWF(n=10)	SWF(n=20)	SWF(n=5)	EKF	iEKF
V1.01	0.24m, 1.46°	0.23m, 1.47°	0.49m, 5.60°	0.71m, 8.50°	0.58m, 2.73°	0.50m, 4.22°	1.52m, 11.60°	1.74m, 10.40°	1.89m, 32.10°
V1.01 TA	0.09m , 2.87°	0.09m , 2.87°	0.20m , 3.62°	0.23m , 3.68°	0.36m , 3.12°	0.36m , 4.72°	1.00m , 8.22°	1.10m , 8.79°	0.04m , 59.10°
V1.02	0.36m, 2.03°	0.36m, 2.04°	-	-	0.93m, 6.25°	1.15m, 6.00°	1.13m, 8.51°	1.24m, 10.01°	0.67m, 7.56°
V1.02 TA	0.15m , 1.52°	0.16m , 1.55°	-	-	0.33m , 3.90°	1.16m , 4.69°	0.30m , 3.68°	0.75m , 8.13°	0.42m , 5.48°
V1.03	1.81m, 9.25°	1.93m, 9.07°	1.56m, 21.23°	1.84m, 23.36°	-	9.54m, 5.41°	1.41m, 3.12°	4.77m , 13.03°	2.33m, 15.92°
V1.03 TA	1.00m , 4.84°	1.12m , 4.90°	0.42m , 6.49°	0.46m , 6.99°	-	6.36m , 25.85°	0.79m , 6.47°	14.84m, 24.58°	1.83m , 8.69°
V2.01	0.22m, 0.77°	0.22m, 0.75°	0.55m, 5.44°	0.56m, 5.98°	1.08m, 4.88°	0.49m, 3.72°	0.37m, 2.81°	1.68m, 5.90°	0.89m, 4.65°
V2.01 TA	0.14m , 0.83°	0.14m , 0.80°	0.45m , 2.88°	0.39m , 2.58°	0.75m , 5.23°	0.23m , 1.71°	0.26m , 1.96°	0.79m , 3.67°	0.48m , 3.58°
V2.02	0.42m, 3.11°	0.42m, 3.19°	1.02m, 12.55°	0.77m, 9.99°	2.26m, 3.17°	1.55m, 16.36°	3.23m, 7.97°	2.02m, 13.56°	0.87m, 3.63°
V2.02 TA	0.24m , 1.67°	0.24m , 1.74°	0.44m , 4.69°	0.38m , 4.13°	0.96m , 3.71°	0.75m , 4.49°	1.42m , 2.99°	0.83m , 4.32°	0.47m , 4.14°

Table 5.1: Root-mean-squared error in translation and rotation on Vicon room sequences from the Euroc MAV dataset. The bold results show the result with the smallest error. The data with TA has trajectory alignment applied to it.

$$\Sigma_{xy} = \frac{1}{n} \sum_{i=1}^n (y_i - \mu_y)(x_i - \mu_x)^T \quad (5.6)$$

Here μ_x and μ_y represent the mean vectors for X and Y . σ_x and σ_y represent the variance vectors for X and Y . Σ_{xy} represent the covariance matrix of X and Y .

Using these values we can define our rotation and translation as:

$$R = USV^T \quad (5.7)$$

$$t = \mu_y - cR\mu_x \quad (5.8)$$

$$c = \frac{1}{\sigma_x^2} \text{tr}(DS) \quad (5.9)$$

Here U and V are obtained from the singular value decomposition of Σ_{xy} as $\Sigma_{xy} = UDV$. Additionally, if $\text{rank}(\Sigma_{xy}) \geq m - 1$, where m is the dimension of the data. We can say

$$S = \begin{cases} I, & \text{if } \det(\Sigma_{xy}) \geq 0 \\ \text{diag}(1, 1, \dots, 1, -1), & \text{if } \det(\Sigma_{xy}) < 0 \end{cases} \quad (5.10)$$

Using the above calculations for the rotation and translations, I was able to improve on the results from [19]. The results can be seen in Table 5.1. From this we can see that trajectory alignment improved majority of the results except for a few. Additionally, there are some rotational errors that are higher compared to the earlier results. When evaluating the results where the error after trajectory alignment was larger (compared to the other tests), it was seen that the error for there trajectories was already large and alignment cause it to grow further.

The experiments run in [19] that are seen in Table 5.1 are listed below:

- MSCKF - Multi-State Constrained Kalman Filter [14] [10] [11]. This algorithm updated the full state with the current IMU data and n past posed with n limited by an upper bound which can be altered to trade-off accuracy and computational speed.

- OKVIS - Open Keyframe Visual-Inertial SLAM [13]. This algorithm updated a sliding window of poses which are deemed important and are called "keyframes". Non-keyframe poses are dropped and keyframes exiting the sliding window are marginalized.
- SWF - Sliding Window smoother, Fixed-Lag Smoother [12]. These smoothers perform similar to MSCKF, but performs Gauss-Newton descent before marginalization to tune the linearization point.
- EKF - Extended Kalman Filter. This algorithm updated the full state with the current position estimates of all features observed and all previous states are marginalized.
- iEKF - Iterated Extended Kalman Filter. This algorithm is similar to EKF except it takes several Gauss-Newton steps before each marginalization step, to tune the linearization point at which marginalization happens.

Chapter 6

Conclusion

Over the period of the thesis, we explored the development of hardware and software that would allow for environment reconstruction on a drone with a ZED2 camera and fisheye camera. The result was great reconstruction from the ZED2 camera. This hardware setup can be used by future for data collection and training to convert the fisheye camera's view to depth information.

The next step for the drone environment reconstruction research is improvement of the structure from motion on the fisheye camera. Additional areas of improvement could be shown by testing the algorithm on a drone. This could also be expanded to run in real-time. This would require transfer of information from the drone to a server as was done in [9].

Near the end of the thesis, we also began an exploration of Dynamic SLAM. This area of research covers performing SLAM in an environment where there are several moving obstacles [21] [2] [7]. There are several possible improvements in dynamic SLAM, such as using estimating acceleration of moving features and using those estimations for better predictions of the moving features.

Bibliography

- [1] J. Bermudez-Cameo, G. Lopez-Nicolas, and J.J. Guerrero. “Line-Images in Cone Mirror Catadioptric Systems”. In: *2014 22nd International Conference on Pattern Recognition*. 2014, pp. 2083–2088. DOI: [10.1109/ICPR.2014.363](https://doi.org/10.1109/ICPR.2014.363).
- [2] Berta Bescos et al. “DynaSLAM: Tracking, Mapping, and Inpainting in Dynamic Scenes”. In: *IEEE Robotics and Automation Letters* 3.4 (Oct. 2018), pp. 4076–4083. DOI: [10.1109/lra.2018.2860039](https://doi.org/10.1109/lra.2018.2860039). URL: <https://doi.org/10.1109%2Flra.2018.2860039>.
- [3] R. Boutteau et al. “Road-line detection and 3D reconstruction using fisheye cameras”. In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. 2013, pp. 1083–1088. DOI: [10.1109/ITSC.2013.6728376](https://doi.org/10.1109/ITSC.2013.6728376).
- [4] James Coughlan and A. Yuille. “The Manhattan World Assumption: Regularities in scene statistics which enable Bayesian inference.” In: *NIPS* (Feb. 1970).
- [5] *Fisheye camera model*. URL: https://docs.opencv.org/3.4.1/db/d58/group_calib3d_fisheye.html.
- [6] Wenliang Gao et al. “Autonomous aerial robot using dual-fisheye cameras”. In: *Journal of Field Robotics* 37.4 (2020), pp. 497–514. DOI: <https://doi.org/10.1002/rob.21946>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21946>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21946>.
- [7] Mina Henein et al. *Dynamic SLAM: The Need For Speed*. 2020. DOI: [10.48550/ARXIV.2002.08584](https://doi.org/10.48550/ARXIV.2002.08584). URL: <https://arxiv.org/abs/2002.08584>.
- [8] *IMX219-200 camera, 200° FOV, applicable for Jetson Nano*. URL: <https://www.waveshare.com/imx219-200-camera.html>.
- [9] Shubh Jagannatha and Nitzan Orr. *Drone-Based Real-Time OpenARK 3D Reconstruction*. Dec. 2020.
- [10] MLA Lourakis and Antonis A Argyros. ““Is Levenberg-Marquardt the most efficient optimization algorithm for implementing bundle adjustment?””. In: *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1. Vol. 2. IEEE*. (2005), pp. 1526–1531.
- [11] Donald W Marquardt. ““An algorithm for least-squares estimation of nonlinear parameters”.” In: *Journal of the society for Industrial and Applied Mathematics* 11.2 (1963), pp. 431–441.

- [12] Anastasios I. Mourikis and Stergios I. Roumeliotis. “A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation”. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation* (2007), pp. 3565–3572.
- [13] Esha Nerurkar, Kejian Wu, and Stergios Roumeliotis. “C-KLAM: Constrained Keyframe-based Localization and Mapping”. In: *Proceedings - IEEE International Conference on Robotics and Automation* (May 2014), pp. 3638–3643.
- [14] Jorge Nocedal and Stephen Wright. “Numerical optimization.” In: *Springer Science Business Media* (2006).
- [15] A. Perez-Yus, G. Lopez-Nicolas, and J.J. Guerrero. “A novel hybrid camera system with depth and fisheye cameras”. In: *2016 23rd International Conference on Pattern Recognition (ICPR)*. 2016, pp. 2789–2794. DOI: [10.1109/ICPR.2016.7900058](https://doi.org/10.1109/ICPR.2016.7900058).
- [16] Alejandro Perez-Yus, Gonzalo Lopez-Nicolas, and Josechu Guerrero. “Peripheral Expansion of Depth Information via Layout Estimation with Fisheye Camera”. In: vol. 9912. Oct. 2016, pp. 396–412. ISBN: 978-3-319-46483-1. DOI: [10.1007/978-3-319-46484-8_24](https://doi.org/10.1007/978-3-319-46484-8_24).
- [17] Alejandro Rituerto, Luis Puig, and J.J. Guerrero. “Visual SLAM with an Omnidirectional Camera”. In: *2010 20th International Conference on Pattern Recognition*. 2010, pp. 348–351. DOI: [10.1109/ICPR.2010.94](https://doi.org/10.1109/ICPR.2010.94).
- [18] Ethan Rublee et al. “ORB: An efficient alternative to SIFT or SURF”. In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. DOI: [10.1109/ICCV.2011.6126544](https://doi.org/10.1109/ICCV.2011.6126544).
- [19] Amay Saxena et al. *Simultaneous Localization and Mapping: Through the Lens of Non-linear Optimization*. Dec. 2021.
- [20] S. Umeyama. “Least-squares estimation of transformation parameters between two point patterns”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.4 (1991), pp. 376–380. DOI: [10.1109/34.88573](https://doi.org/10.1109/34.88573).
- [21] Jun Zhang et al. *VDO-SLAM: A Visual Dynamic Object-aware SLAM System*. 2020. DOI: [10.48550/ARXIV.2005.11052](https://doi.org/10.48550/ARXIV.2005.11052). URL: <https://arxiv.org/abs/2005.11052>.