THE UNIVERSITY OF TEXAS AT AUSTIN

UNDERGRADUATE HONORS THESIS

Exploiting a Prior 3D Map for Object Recognition

Author:

Siddarth KAKI

Supervisors:

Dr. Todd HUMPHREYS

Dr. Maruthi AKELLA

Bachelor of Science in Aerospace Engineering

Cockrell School of Engineering

The University of Texas at Austin

May 2018

THE UNIVERSITY OF TEXAS AT AUSTIN

Abstract

Dr. Todd Humphreys

Dr. Maruthi Akella

Cockrell School of Engineering

Bachelor of Science in Aerospace Engineering

Exploiting a Prior 3D Map for Object Recognition

by Siddarth KAKI

Recognizing objects in the environment and precisely determining their positions is a fundamental component of autonomous navigation systems. This thesis presents a technique for determining both the locations and the semantic labels of new objects in a scene with respect to a prior three-dimensional (3D) map of the scene. This work aims to reduce object recognition errors in cluttered environments by isolating new objects from the known background by correlating features detected in a new photo with feature points that constitute the 3D map. Such isolation enables a neural network trained to recognize an enumerated set of objects to focus narrowly upon those portions of images that contain new objects instead of having to process the whole scene. As a result, changes in a prior map can be rapidly detected and semantically labeled, allowing for confident navigation within the ever-evolving cluttered environment. Using multiple images obtained from varying camera poses, the globally-referenced 3D positions of changes in the scene can be determined with multiple-view geometry techniques.

Acknowledgements

This project would not have been possible without the guidance, support, and advice from my two faculty advisors, Dr. Todd Humphreys and Dr. Maruthi Akella. I have been working with Dr. Humphreys since my freshman year. I have grown significantly as a student and as a researcher over these past four years under Dr. Humphreys. My involvement with Dr. Humphreys' lab has provided me outstanding exposure to the various aspects of autonomous robotics. Dr. Akella has been a tremendous mentor over the past few years as well, and has been exceptionally helpful in terms of personal and academic advice. I have thoroughly enjoyed working under Dr. Humphreys and Dr. Akella.

I would also like to extend my gratitude to several graduate students. Tucker Haydon has played a critical role in my work, providing me the data I needed, and offering advice and help on countless occasions for 3D reconstruction. Marcelino Almeida and Nick Montalbano have been dependable resources to bounce ideas off of. Lakshay Narula has been my go-to resource for computer vision questions. I would also like to offer my thanks to my fellow undergraduate student Vatsa Gandhi, who has contributed tremendously to the design of our quadrotor vehicle platform.

I would like to conclude with offering my thanks to Megan McFadden for her guidance in navigating through the engineering honors thesis process, and to my family and friends for supporting me. This thesis would not be possible without the involvement of everyone mentioned, and I offer my deepest gratitude to them.

Contents

List of Figures

List of Abbreviations

3D	Three-**D**imensional
ARENA	**A**utonomous **R**obotics **E**xploration **N**etted **A**rena
AWS	**A**mazon **W**eb **S**ervices
CDGNSS	**C**arrier-phase **D**ifferential **G**lobal **N**avigation **S**atellite **S**ystem
CNN	**C**onvolutional **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
DBSCAN	**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise
DSP	**D**igital **S**ignal **P**rocessor
FLANN	**F**ast **L**ibrary for **A**pproximate **N**earest **N**eighbors
FPS	**F**rames-**p**er-**S**econd
GB	**G**iga**b**yte
GNSS	**G**lobal **N**avigation **S**atellite **S**ystem
GPS	**G**lobal **P**ositioning **S**ystem
GPU	**G**raphics **P**rocessing **U**nit
IMU	**I**nertial **M**easurement **U**nit
k-NN	**k**-**N**earest **N**eighbors
MAV	**M**icro **A**erial **V**ehicle
RAM	**R**andom **A**ccess **M**emory
RF	**R**adio **F**requency
SfM	**S**tructure-**f**rom-**M**otion
SIFT	**S**cale **I**nvariant **F**eature **T**ransform
SLAM	**S**imultaneous **L**ocalization **A**nd **M**apping
SOC	**S**ystem-**o**n-a-**C**hip

Chapter 1

Introduction

1.1 Motivation

High-definition 3D maps offer tremendous benefits to autonomous vehicles, from reducing difficulty of parsing environments to enabling precise localization in GNSS-challenged environments [1], [2]. Such maps enable object detection and tracking algorithms to "expect the expected"; detected features of a house, tree, or statue may be correlated to features present within the map. Thus, the algorithm immediately recognizes these objects as part of the static background. A broader research area the Radionavigation Laboratory at The University of Texas at Austin is pursuing is the digital 3D reconstruction of environments from photos with structure-from-motion (SfM) techniques. Photos are taken with CDGNSS-enabled quadrotor aerial vehicles able to localize themselves to centimeter accuracy. By employing SfM algorithms and the precise pose of the quadrotor vehicle when it takes each photo, a globally-referenced digital 3D map is constructed. Maps generated with the aid of GNSS may even substitute for GNSS during GNSS-challenged situations such as signal blockage or jamming [3].

However, such maps have a major drawback; they are simply static single-instance-in-time snapshots of the constantly-changing world. Static maps, while tremendously valuable, pose serious safety risks to automated vehicles themselves, living beings, and public infrastructure whenever the maps go out-of-date. A primary obstacle to constantly taking new photos of previously mapped regions and incorporating the new photos to update the map is that the map generation

is a computationally expensive process. Consider the reconstruction for a structure dubbed the "chimney" depicted in Fig. 1.1, which measures approximately 90 centimeters in height and 60 centimeters in width on each side. The reconstruction process is run on a p2.xlarge Amazon Web Services (AWS) cloud instance [1]. Even with the significant computational resources the AWS instance provides, the "chimney" reconstruction takes approximately 12 hours to complete. Naïvely updating a map with a constant stream of new images is infeasible, both in terms of computational resources and time available. In addition, not all changes that occur in the real world are necessarily important enough to reflect in a map. For example, people and cars always move, and thus should not be considered to be a part of the static environment the map is intended to represent. There exists a need to accurately and efficiently identify changes in the real-world environment with respect to *a priori* generated 3D maps that prove significant enough to necessitate updating the maps.

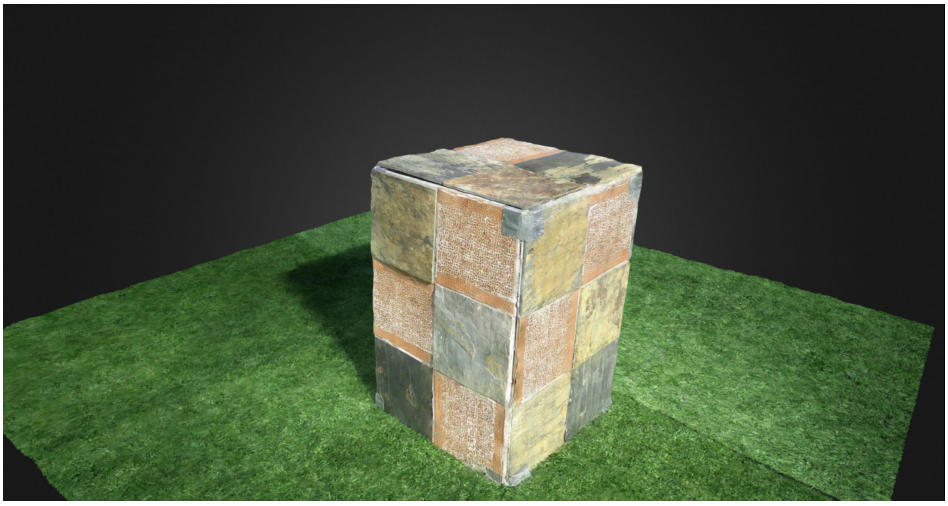


FIGURE 1.1: Reconstruction of the "Chimney"

[1] https://aws.amazon.com/ec2/instance-types/p2/

1.2 Background

The Radionavigation Laboratory is actively conducting research to develop high-definition, globally-referenced, to-scale, 3D maps. A quadrotor micro aerial vehicle (MAV) platform (see Fig. 1.2) has been developed in-house to carry out mapping efforts. The MAVs are equipped with centimeter-accurate carrier-phase differential GNSS (CDGNSS) -based positioning sensors. Each MAV is equipped with a 4K-resolution camera. 3D reconstruction is performed with the following as inputs: 1) high-resolution photos of regions-of-interest to map, and 2) accurate knowledge of the camera pose associated with each photo.

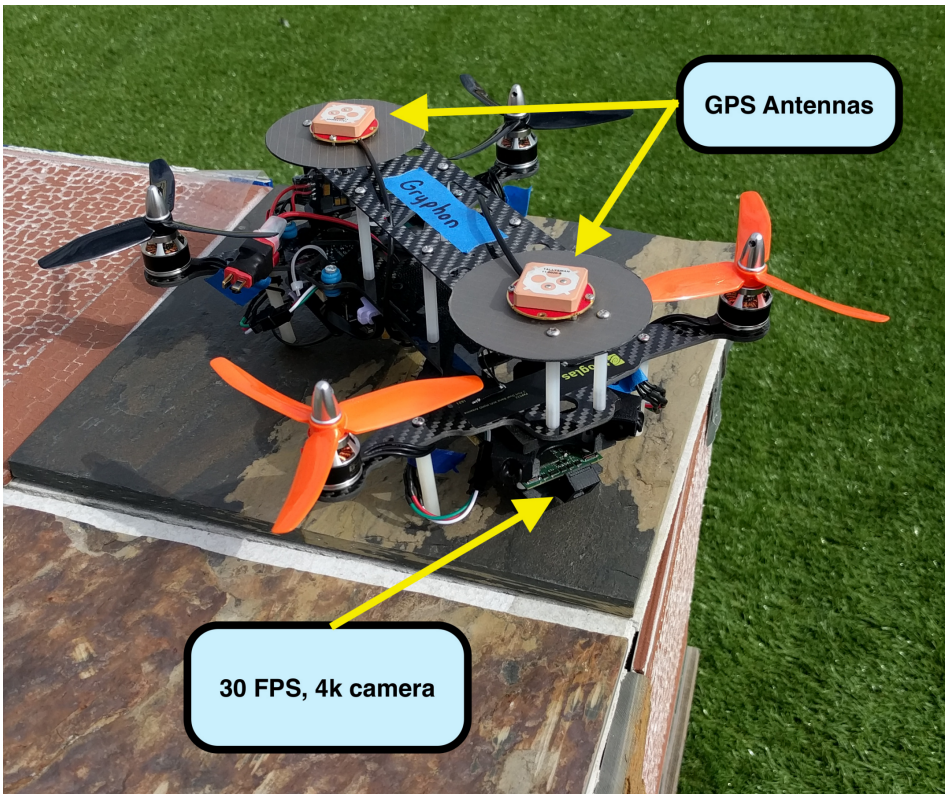


FIGURE 1.2: Custom Quadrotor MAV

1.2.1 CDGNSS for Mobile Applications

The MAVs developed for mapping are equipped with CDGNSS-based positioning. The low-cost CDGNSS implementation for mobile platforms has been developed in-house, based off a dense

reference network for mass-market centimeter-accurate positioning [4], [5]. These advances in low-cost centimeter-accurate positioning has enabled implementation on small MAVs.

1.2.2 Image Features

Image features are important for computer vision applications. A feature is essentially some small patch of an image that contains some unique characteristic. A feature might represent some structure in the image, such as a point, an edge, an area of high contrast, etc. Each feature consists of two major components: 1) the feature keypoint, and 2) the feature descriptor. The feature keypoint (or just feature point) primarily contains the position of the feature in the image. The feature descriptor is the mathematical representation of the feature. Having mathematical representations for features allows for feature comparison. For example, two features from two different images can be easily compared to determine whether they represent the same part of the physical object of which the images were taken.

Several feature extraction algorithms exist in the literature. This work employs the scale-invariant feature transform (SIFT) algorithm [6].

1.2.3 Camera Parameters

Cameras are often modeled as pinhole cameras [7]. The camera model is expressed by so-called "intrinsic parameters" (or just "intrinsics"). These parameters describe the focal length and image center of the camera. Camera distortion coefficients are also important camera-specific parameters.

The camera "extrinsic parameters" (or just "extrinsics") refer to the pose of the camera in the world frame (which is the globally-referenced frame in this work).

1.2.4 3D Mapping

Standard techniques exist for 3D reconstruction from photos. In computer vision literature, these techniques are classified as structure-from-motion (SfM) algorithms. In robotics literature, these techniques are classified under the term simultaneous localization and mapping (SLAM). In either

case, the fundamental underlying optimization technique is known as bundle adjustment. Bundle adjustment fundamentally works by determining the positions of feature points detected in the environment from multiple images with respect to the images themselves, simultaneously solving for the relative positions of the environmental feature points and the relative pose of the camera when it took each image. Fig. 1.1 depicts the textured dense reconstruction of the "chimney" structure.

While globally-referenced camera pose priors are not required for bundle adjustment, they offer two important benefits. One, having global-references for the camera poses in the relative geometry enables solving for global-references for each of the environmental feature points. For navigation purposes, globally-referenced maps offer many advantages, especially when sensors (such as radar, cameras, or LIDAR) are functioning sub-optimally. When other sensors are challenged, having a globally-referenced map offers a valuable fall-back option. In general, such maps bolster the robustness for autonomous navigation. Two, global-referencing constrains the geometry of the reconstruction to be metric-accurate; that is, a distance of 1-meter as measured in the map will indeed represent a 1-meter distance in the real environment the map represents. Having a to-scale map is essential for navigation. In summary, globally-referenced camera pose priors enable the generated maps to be globally-referenced, and to-scale.

1.2.5 Object Detection and Recognition

Accurate object detection and classification have been greatly sought-after research goals. While object recognition accuracy with machine learning has improved vastly over the past few years [8], such systems still struggle to accurately parse cluttered or noisy (rapidly evolving) scenes. Current techniques for object recognition in cluttered environments employ semantic instance segmentation [9] to first segment an image into unidentified objects, and then pass the resulting segments to a neural network for object recognition, as opposed to attempting to classify the image as a whole. These techniques can achieve high recognition accuracy for some scenes, but can fail catastrophically for others – especially scenes of cluttered environments.

Convolutional neural networks (CNNs) have proven to be the most apt class of machine learning architectures for image classification [8], [10]. Several CNN architectures (such as MobileNet [11] and Inception-v3 [12]) exist, varying in performance metrics such as mean average precision, memory and computing footprint, and real-time performance. While Inception-v3 currently is state-of-the-art in terms minimal error rate, the architecture requires significant computing prowess, rendering real-time operation on a mobile platform infeasible. Though not as accurate as Inception-v3, the MobileNet class of architectures is designed to run real-time on a smartphone, making MobileNet ideal for real-time processing on a MAV.

Chapter 2

Experimental Setup

2.1 Quadrotor Platform

A custom quadrotor MAV platform has been developed for research application purposes (see Figs. 1.2, 2.1). Some parts are purchased off-the-shelf, while others are designed and built in-house. Many interconnected parts compose a MAV, and only a few unique and relevant aspects are discussed.

FIGURE 2.1: Custom Quadrotor MAV

2.1.1 Snapdragon Flight

The primary electronics package on-board each of these MAVs is the Qualcomm Snapdragon Flight board (see Fig. 2.2). The primary component of the Snapdragon Flight board is the Snadragon 801 system-on-a-chip (SOC), which contains a quad-core Krait CPU, a Hexagon DSP, an Adreno GPU, and 2 GB of RAM. In addition, the Snapdragon Flight board includes a Sony IMX135 4K-resolution camera, and an inertial measurement unit (IMU). The SOC runs the low-level PX4 flight software stack. In-house developed high-level controllers run on the SOC, interfacing with the low-level PX4 controllers.

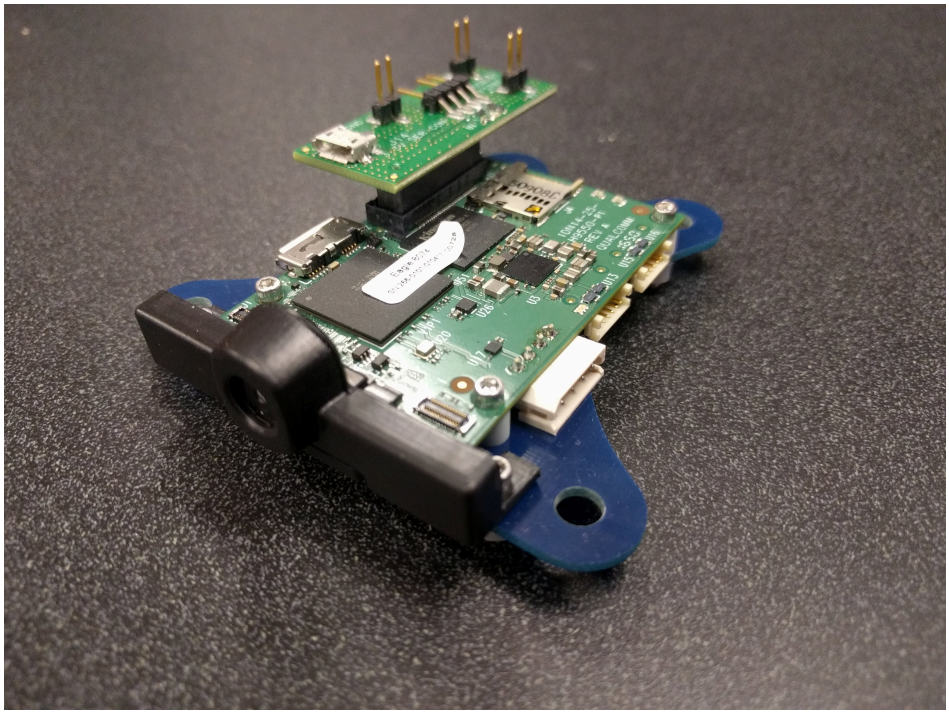


FIGURE 2.2: Snapdragon Flight Board

2.1.2 ODROID-XU4

Each MAV is equipped with an ODROID-XU4 mobile processing board (see Fig. 2.3). The XU4 module runs the software-defined radio GNSS-based pose solutions for the MAV.

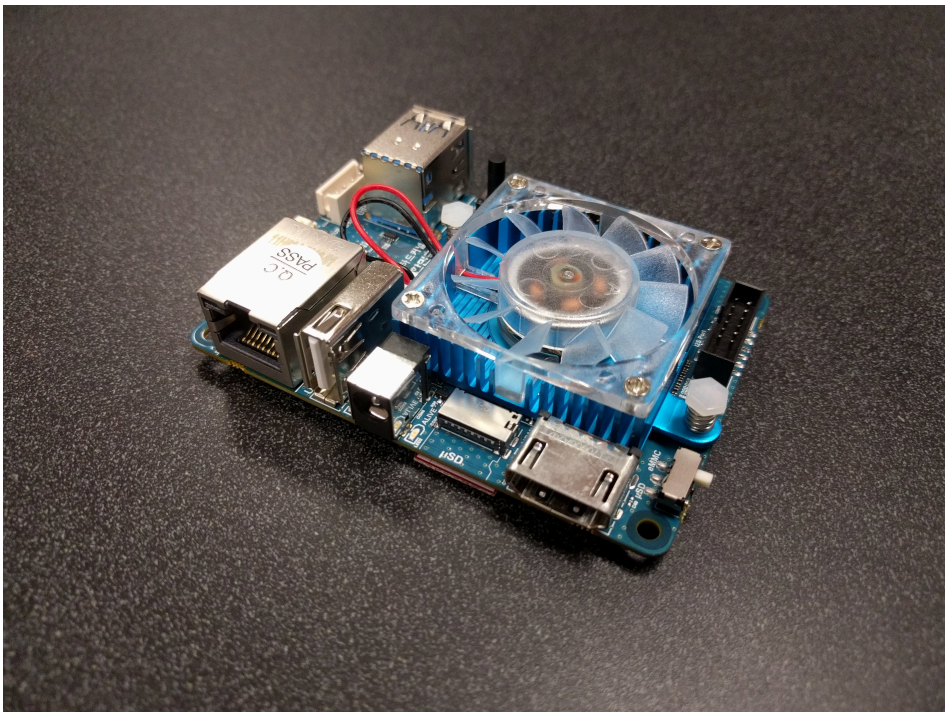


FIGURE 2.3: ODROID-XU4 Board

2.1.3 RadioLynx

Each MAV is equipped with a RadioLynx radio-frequency (RF) front-end board (see Fig. 2.4). The RadioLynx module performs analog-to-digital conversion of the GNSS signals, before passing them onto the ODROID to generate GNSS-based pose solutions. The RadioLynx is a dual-frequency, dual-antenna input device.

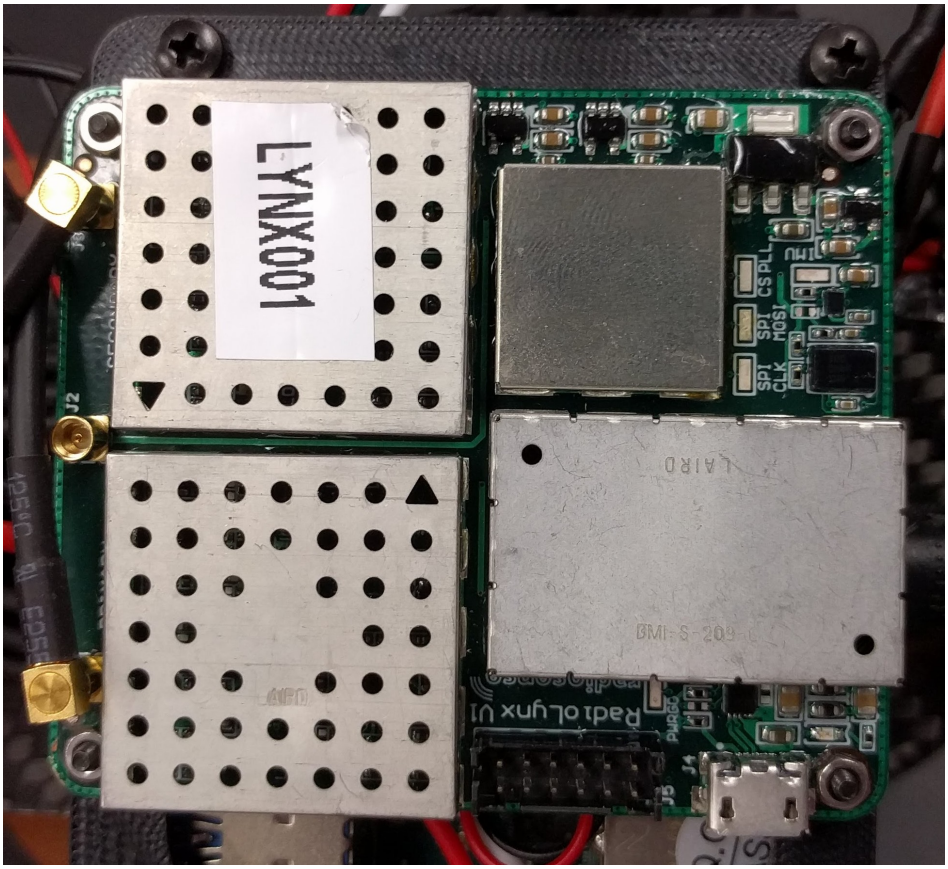


FIGURE 2.4: RadioLynx Board

2.2 ARENA

Test flights and research flight campaigns are conducted in an outdoor rooftop flight facility dubbed the Autonomous Robotics Exploration Netted Arena (ARENA) (see Fig. 2.5).

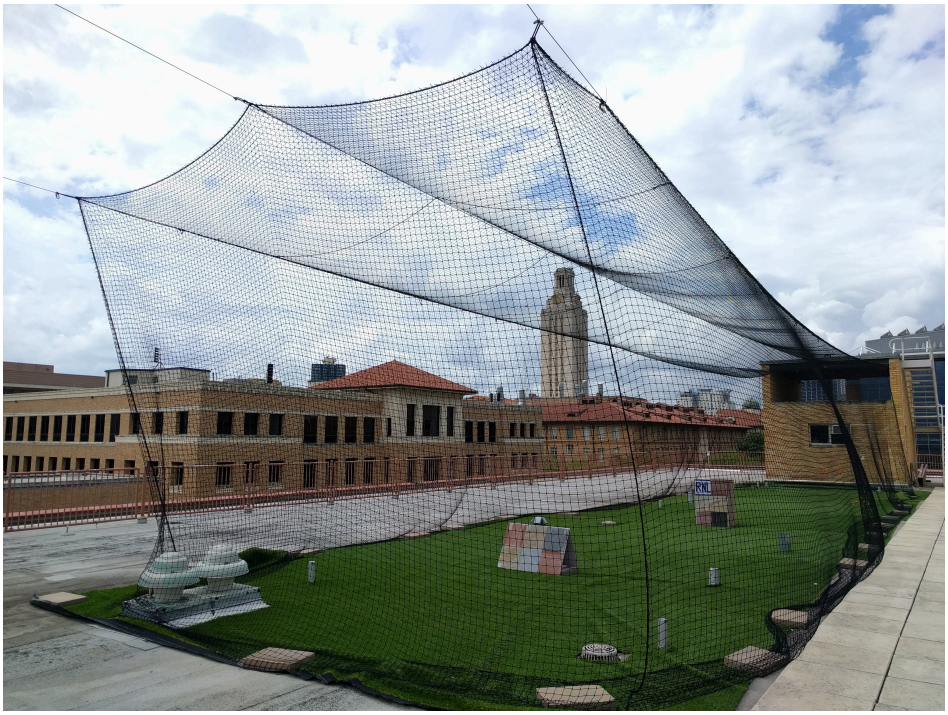


FIGURE 2.5: ARENA Facility

2.2.1 Flight Environment

The ARENA consists of a flight environment roughly measuring 20 meters in length, by 5 meters in width, by 3.5 meters in height. The flight area is encompassed by a net for safety purposes. The ground is covered with artificial turf to protect the MAVs in the event of a crash.

2.2.2 The "Chimney"

An artificial structure composed of flooring tiles, dubbed the "chimney", has been constructed to serve as the test environmental object for 3D mapping (see Fig. 2.6). The "chimney" is in the

middle of the ARENA, and measures approximately 90 centimeters in height and 60 centimeters in width on each side. The surface of the "chimney" provides ample, distinct feature points, which is a necessity for high-quality 3D reconstruction.

FIGURE 2.6: The "Chimney"

Chapter 3

Software Design

3.1 Software Dependencies

This work depends on several open-source software packages, some of which are described below.

3.1.1 COLMAP

COLMAP [13], an open-source SfM algorithm, is used as a foundation for integrating globally-referenced camera pose priors. COLMAP produces several outputs in the form of text files and SQLite databases. This work solely uses COLMAP's sparse point cloud (sans surfaces and texturing) output data products, which encompasses the following files:

- `cameras.txt`: text file containing the camera intrinsics for each unique camera that has taken images used in the reconstruction
- `images.txt`: text file containing the pose and feature points for images used in the reconstruction
- `points3D.txt`: text file containing, for each sparse cloud feature point, 1) the 3D position, and 2) indices indicating the locations in `database.db` where the feature descriptors for each feature point are stored
- `database.db`: SQLite database containing, among other data, information about each feature point and the corresponding descriptor

For a more comprehensive description of the COLMAP output data products, refer to the COLMAP documentation [1].

3.1.2 OpenCV

OpenCV is a popular open-source computer vision library written in C/C++ [14]. OpenCV includes implementations for various computer vision algorithms employed in this work.

3.1.3 Eigen

Eigen is a popular C++ template library for linear algebra [15]. Eigen is heavily used for matrix manipulation and operations in this work.

3.2 Algorithm Design

The proposed algorithm and implementation to efficiently determine when and where to take new photos for map updates is outlined as follows:

1. Fly a MAV through a previously mapped environment
2. Strategically capture a photo of the environment
3. Perform SIFT feature extraction on this image
4. Project the MAV into the sparse point cloud map of the scene
5. Capture a virtual "photo" of the scene, consisting of the feature points from the map
6. Correlate feature descriptors of the real photo with those of the "virtual photo"
7. Isolate uncorrelated features (which represent any changes in the environment with respect to the map, in addition to noise)

[1] https://colmap.github.io/index.html

8. Perform clustering of the uncorrelated features to isolate the features that likely belong to a significant change in the environment (as opposed to just noise)
9. Determine the minimum bounding box around the isolated features from the previous step
10. Crop the real photo down to the bounding box to isolate the image to only the significant change in the environment
11. Parse the cropped photo with a CNN trained to detect and recognize objects to determine what the change in the environment actually is
12. Determine whether the change in the environment is worthy of updating the maps with heuristics or machine learning
    (a) If remapping is determined to be necessary, fly the MAV around the region and take new photos, while tracking the feature points of the environmental change
    (b) With a sufficient number of images, determine the 3D globally-referenced position of the change in the environment with multiple-view geometry techniques
    (c) Conduct a map update campaign at the location determined in the previous sub-step

This work implements a subset (steps 1-10) of the above proposed algorithm in C++. The remaining steps are future work. Significant aspects of the portions implemented in this work are further detailed in the following sections.

3.2.1 Loading the Map

The map is represented by the sparse 3D point cloud, which includes the feature points themselves, and the corresponding feature descriptors. The map is loaded from `points3D.txt` and `database.db` into a C++ object, which takes approximately 10-15 seconds for the "chimney". To speed up map loading, a serialization routine is run to store only the relevant components of the two files into a separate text file. Loading and deserializing from the serialized map file takes less than one second.

3.2.2 SIFT Feature Extraction

When a new image is taken, SIFT features are extracted from that image via a COLMAP command line utility. These features and corresponding feature descriptors are read into OpenCV matrix objects. The COLMAP SIFT utility itself is built around the VLFeat implementation [16] of the original SIFT algorithm [6].

3.2.3 Map Projection

For each new image taken, the MAV is placed and oriented within the sparse point cloud using its pose when the image was taken. Based off these camera extrinsics and the camera intrinsics from calibration, the 3D point cloud is projected onto the virtual camera image plane, producing the virtual "photo."

3.2.4 Feature Correlation

Once the expected features (via the map projection) and the actually seen features (from feature extraction of the new image) are available, the two sets of features are correlated to determine which features in the new image exist in the map, and which features do not match (which would include any changes in the environment, plus noise). Several techniques are employed in the feature correlation routine for efficiency and robustness:

3.2.4.1 Directed Search

A brute-force matching method would compare each feature descriptor with every single other descriptor without considering the locations of the feature keypoints, an expensive and time-consuming process. However, because the MAV's pose with respect to the map is known to high accuracy due to global-referencing for both the MAV and the map, the region where a certain feature would be expected to be found, if a match exists, can be limited in scope from the entire image down to a circle of small radius (choice depends on the uncertainty of the knowledge of the MAV's pose) around the keypoint pixel location. That is, if a match were to exist for a specific

feature keypoint, it should exist in approximately the same two-dimensional pixel location as the original keypoint, with some small uncertainty due to uncertainty in the MAV's pose solution. The number of features to compare against is significantly reduced by this process, which is known as "directed search." In addition, directed search makes the algorithm more robust because by restricting the search space to a narrow area, the chance of incorrectly correlating to an incorrect match is significantly reduced.

3.2.4.2 Ratio Test

The OpenCV implementation for the Fast Library for Approximate Nearest Neighbors (FLANN) [17] is employed for matching. The FLANN implementation is configured to provided the two closest matches for a feature point within its directed search region. The metric for matching similarity is the Euclidean distance between two feature descriptor vectors. The ratio of the best match distance to the second-best match distance is taken. If the radio is greater than a configurable parameter, then the keypoint is considered a match. The ratio test essentially ensures a higher probability of correct matches because only feature points with best matches that are significantly better than the second-best matches are considered to actually be matches. Feature points not matched are isolated.

3.2.5 Clustering

The set of unmatched feature points represents one or both of the follow two parts: 1) a change in the environment (which is sought), and 2) extraneous noise (which is unwanted). The feature points from a change in the environment must be isolated from the feature points that represent noise. It is assumed that a change in the environment, such as a new object, consists of a high density of feature points, while feature points due to noise are randomly distributed at a lower density. With this assumption, the set of unmatched feature points is parsed with a clustering algorithm, which attempts to find high-density regions of feature points. For clustering, the density-based spatial clustering of applications with noise (DBSCAN) [18] algorithm is employed. DBSCAN is preferred to other clustering algorithms such as k-nearest neighbors (k-NN) because DBSCAN

does not need an input for how many clusters to look for, unlike k-NN; this design enables DBSCAN to find as many clusters that meet its configurable threshold. On the other hand, k-NN requires user input for how many clusters to look for. Because the number of new objects introduced to an environment is not known beforehand, k-NN is not useful, while DBSCAN is desirable.

Assuming DBSCAN successfully finds the cluster(s) of feature points corresponding to changes in the environment, a minimum bounding box for each cluster is determined. The actual new image captured is cropped down to the bounding box, thus isolating the change in the environment from the rest of the image.

3.2.6 Object Recognition

The cropped image is parsed by a deep convolutional neural network (CNN) trained to recognize objects. The MobileNet CNN architecture [11] implemented in TensorFlow [19] is chosen. The MobileNet class of architectures are designed for running on mobile platforms, which is ideal for this use-case because the MAV's Snapdragon processor is heavily based off a mobile smartphone processor. The CNN classifies the change in the environment provided to it in the form of the cropped image. Based off this knowledge, it can be determined whether updating the map is desired based of heuristics or a machine learning model.

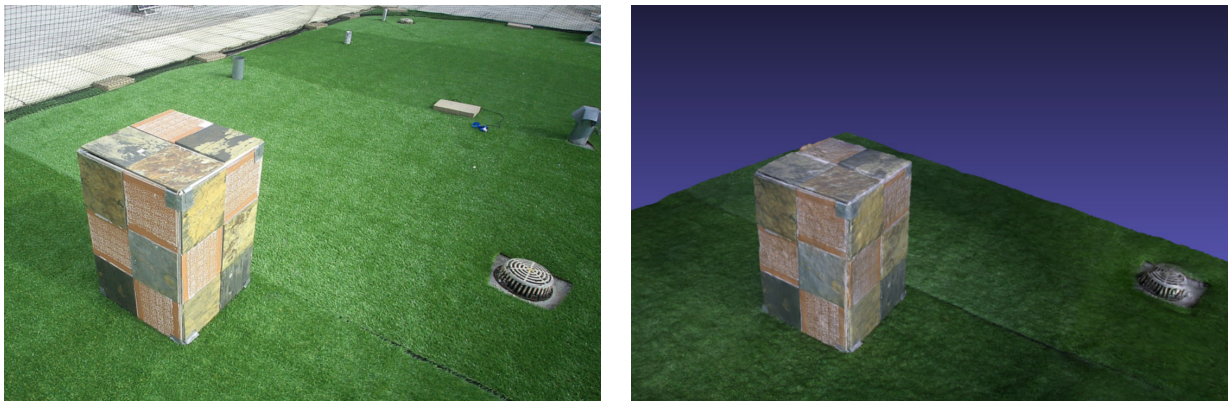
Chapter 4

Simulated Results

4.1 Current State of Work

A nearly end-to-end version of the algorithm described in Chapter 3 has been implemented in C++. However, there exists a significant amount of noise in the system, especially due to SIFT's lack of robustness against changing environmental conditions (such as lighting). The algorithm is still being improved to address the noise issues in SIFT, and thus real experimental results must be deferred to future work.

However, with a fusion of real and simulated data as inputs, a slightly simplified version of the algorithm implemented in MATLAB produces satisfactory results as a proof-of-concept. The performance of this MATLAB implementation is described and analyzed. For these results, the mapped environment in consideration is the "chimney" described previously. The "chimney" is depicted in Fig. 4.1, where Fig. 4.1a shows a real photo, while Fig. 4.1b shows a textured dense reconstruction viewed from the same vantage as for the real photo.



(A) Real Photo (B) Reconstruction

FIGURE 4.1: The "Chimney": Real and Reconstructed

4.2 Prior Sparse Map

The prior point cloud representing the map is loaded. The feature keypoints used are the actual keypoints from the map, but the corresponding feature descriptors are simulated. Simulating the descriptors avoids the issue of SIFT's lack of robustness and provides control over the performance of the algorithm, which is important for analyzing how the algorithm responds to varying magnitudes of noise. The 3D point cloud is visualized as viewed from an arbitrary pose in Fig. 4.2.

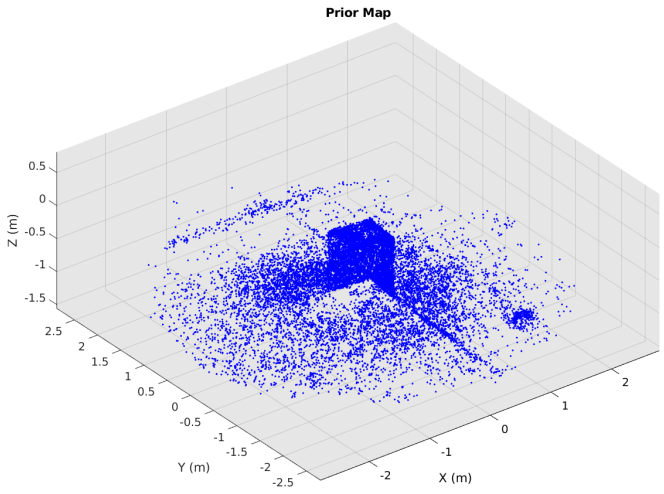


FIGURE 4.2: Prior Sparse Map

4.3 Inserting a New Object

In order to detect a new object in the environment, a new object must first exist. To that end, a virtual sphere of radius 0.25 meters is inserted into the sparse map, as shown in red in Fig. 4.3. This sparse point cloud with the inserted object serves as the ground truth. The virtual sphere is simulated by generating feature keypoints in the shape of a sphere, and generating corresponding descriptors. This virtual sphere acts as a proxy for a small box that is actually inserted into the environment, as shown in Fig. 4.4.

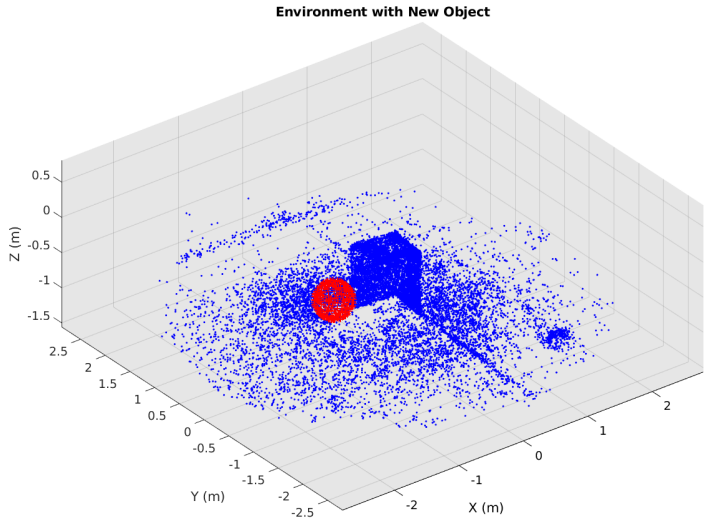


FIGURE 4.3: Prior Sparse Map with Inserted Object

FIGURE 4.4: Environment with New Object

When a real photo of the environment is taken (Fig. 4.4), the MAV is placed within the prior map based off its current pose, and a virtual "photo" is taken. That is, the 3D point cloud representing the prior map is projected onto the MAV's virtual image plane (Fig. 4.5), resulting in the features that the MAV expects to see from its current pose.

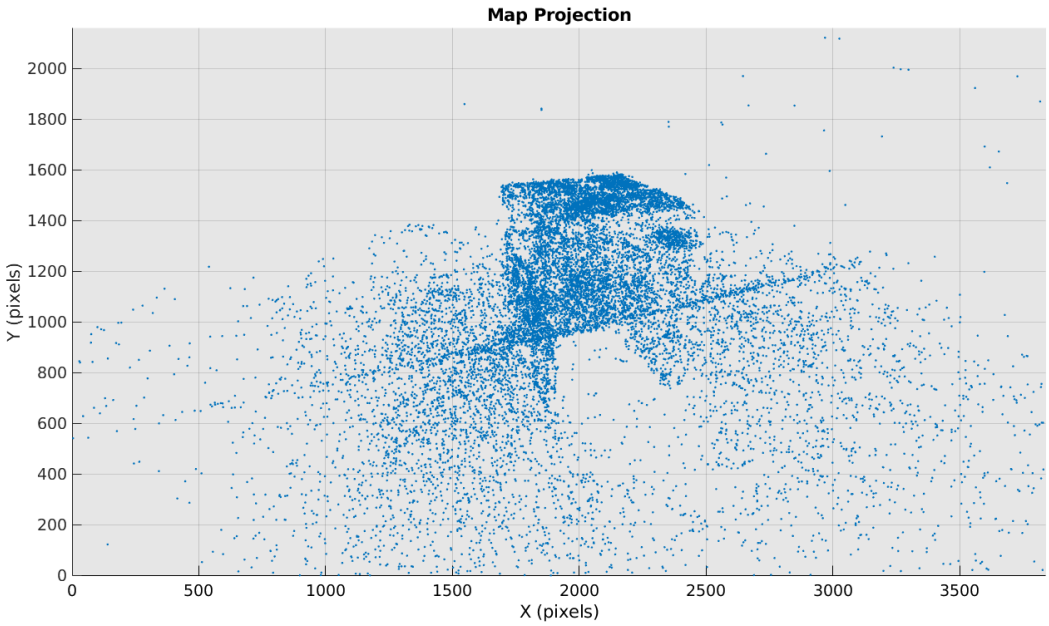


FIGURE 4.5: Map Projection

4.4 Simulating the Camera View

Because the descriptors for the prior map have been simulated (and are not the true descriptors), simply extracting the descriptors from the photo in Fig. 4.4 will not provide the correct descriptors. Thus, the features extracted must be simulated. The point cloud with the new object is tweaked in the following fashion to generate a simulated camera view:

- 2000 randomly generated feature points are added
- 10000 feature points are randomly deleted
- Gaussian random noise with $\mathcal{N}(0\,cm,\ 1\,cm)$ is added to the locations of the remaining feature points

- Uniform random noise sampled from the range $[-20, 20]$ is added to each of the elements for each of the SIFT descriptor vectors

These changes reflect what might be expected if feature points were to be extracted from the real photo in Fig. 4.4 given uncertainty in the MAV's pose and noise due to SIFT. The adjusted point cloud is projected onto the image plane, resulting in the simulated camera view in Fig. 4.6.

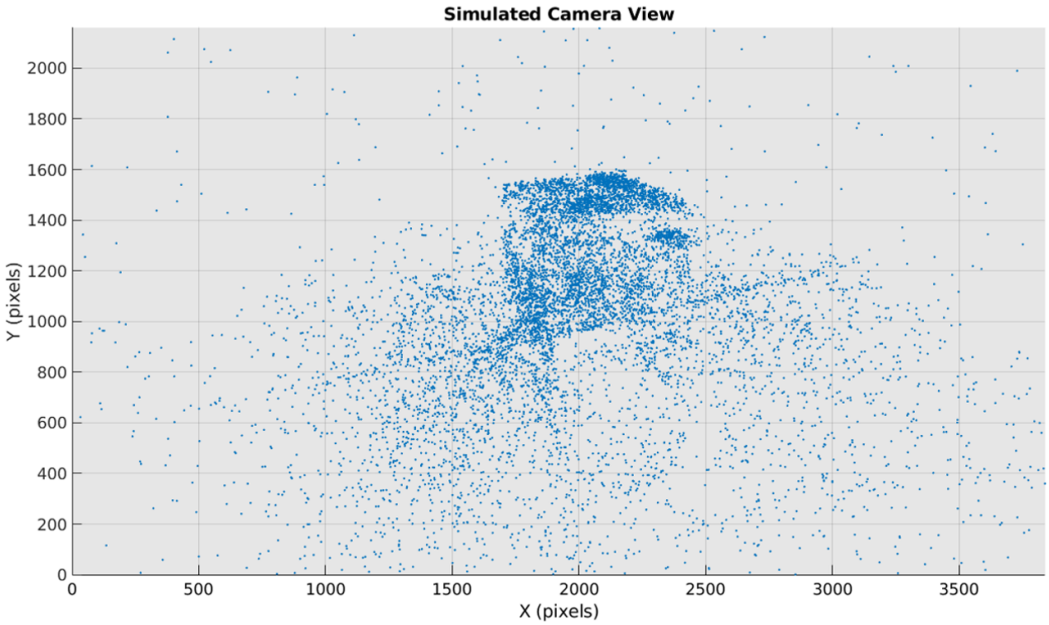


FIGURE 4.6: Simulated Camera View

4.5 Feature Correlation

Two sets of features are present now: 1) the expected features as shown in Fig. 4.5, and 2) the features actually visible as shown in Fig. 4.6. Finding matches between these two sets is attempted in order to determine which feature points that are currently visible correspond to feature points that have existed before; that is, feature points that are expected to be seen. Any unmatched

features correspond to either a change in the environment (such as a new object), or noise. The results of feature correlation are depicted in Fig. 4.7. The blue points represent matched features from the camera view, while the red points represent unmatched features. It is obvious from inspection which feature points correspond to the inserted sphere, and which correspond to noise.

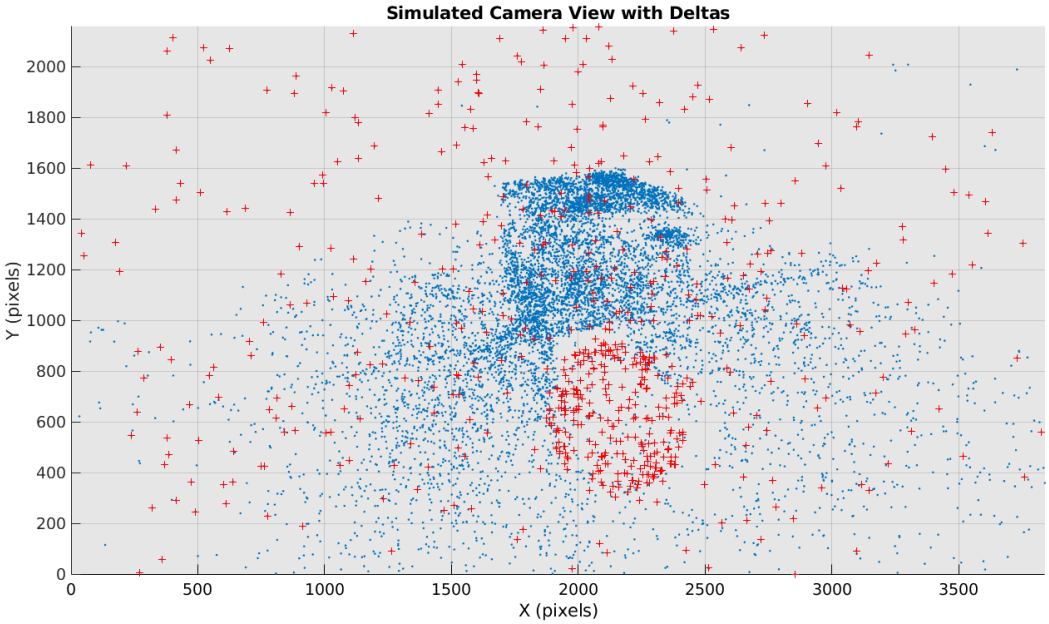


FIGURE 4.7: Simulated Camera View with Deltas

4.6 Clustering the Deltas

From the set of unmatched feature points, the feature points corresponding to the new object must be isolated from the noise. To this end, the set of unmatched feature points is parsed with the DBSCAN clustering algorithm. The results are depicted in Fig. 4.8. The red points represent the cluster corresponding to the new object, while the black points correspond to noise.

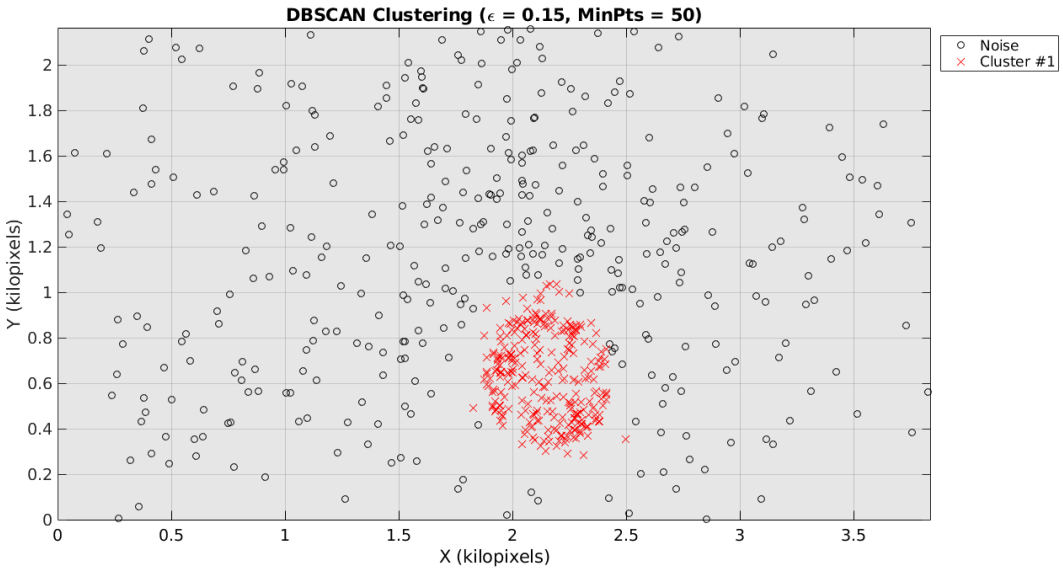


FIGURE 4.8: Clustering of Unmatched Feature Points

A bounding box for the cluster representing the new object is generated, as shown with the red box in Fig. 4.9.

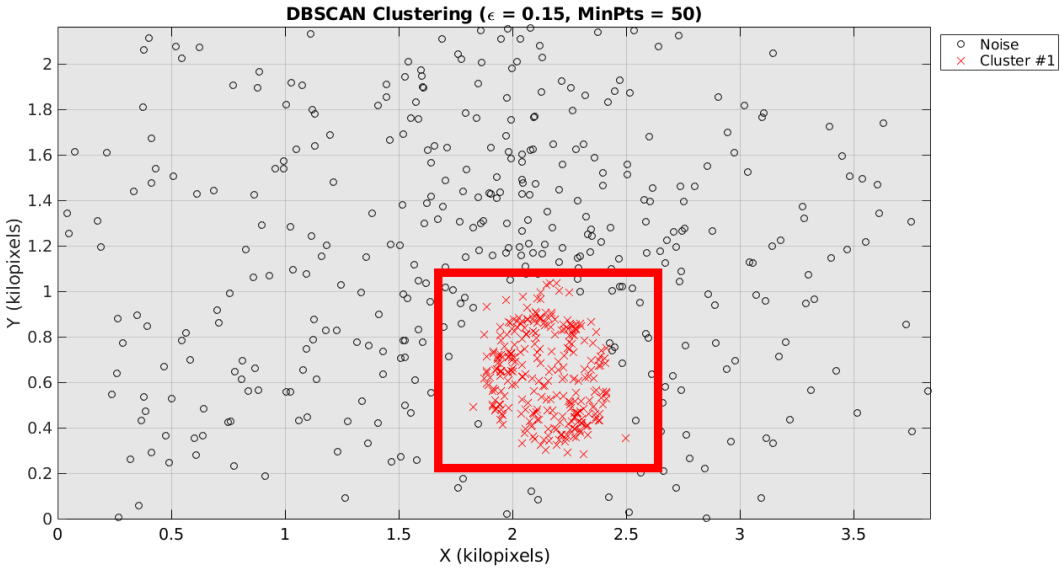


FIGURE 4.9: Clustering of Unmatched Feature Points with Bounding Box

4.7 Cropping and Classifying the Real Photo

Once the bounding box is generated, the real photo is cropped down to the bounded region, as shown in Fig. 4.10. This cropped photo is parsed by the MobileNet CNN to determine the object.

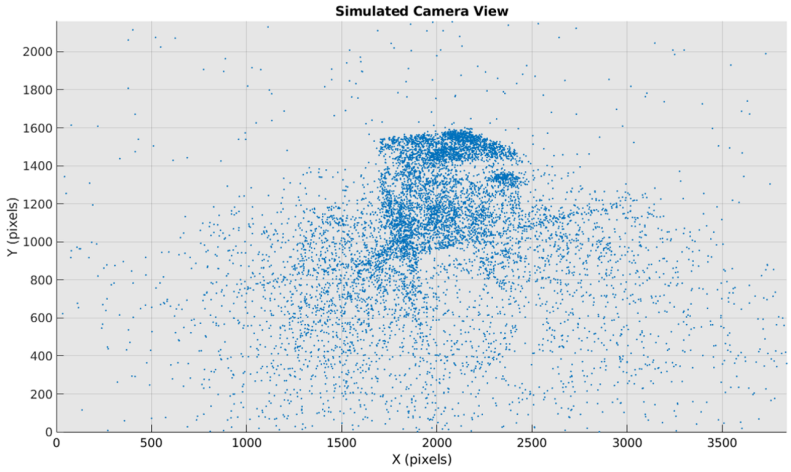
FIGURE 4.10: Real Photo with Bounding Box

4.8 Effect of Noise

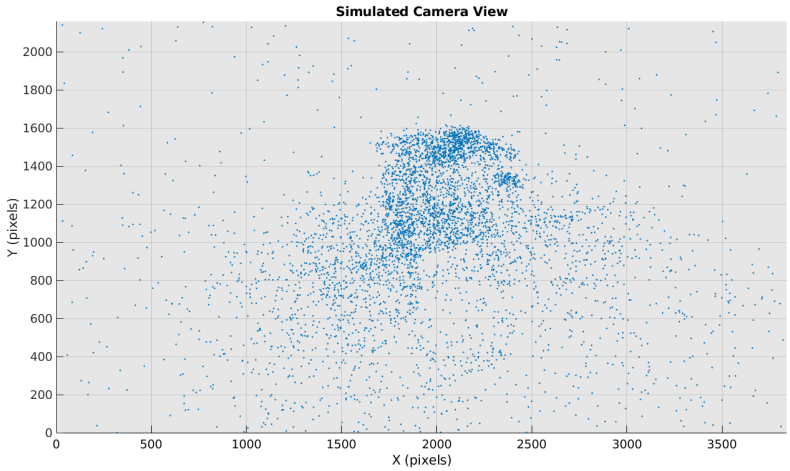
The effect of noise in the SIFT keypoints and descriptors is analyzed with simulated results. Consider the simulation of the camera view described in Sec. 4.4. The point cloud with the new object is tweaked in the following fashion to produce a more noisy simulated camera view than that in Sec. 4.4:

- 5000 randomly generated feature points are added
- 17500 feature points are randomly deleted
- Gaussian random noise with $\mathcal{N}(0\,cm,\,2\,cm)$ is added to the locations of the remaining feature points
- Uniform random noise sampled from the range $[-25, 25]$ is added to each of the elements for each of the SIFT descriptor vectors

The simulated camera view in Sec. 4.4 and subsequent steps are referred to as "Trial 1" hereafter. The above described simulation and subsequent steps are referred to as "Trial 2" hereafter. The simulated camera views for the two trials are depicted in Fig. 4.11. The "Trial 2" view is visibly more noisy and less defined, and more akin to a realistic feature point distribution than the "Trial 1" view.



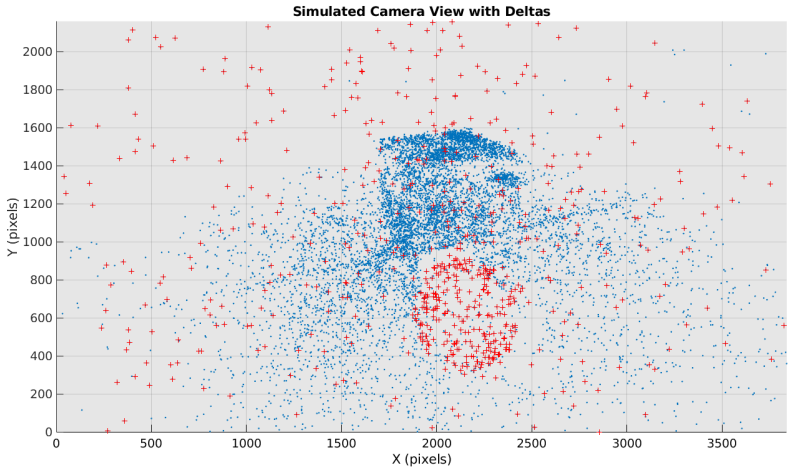
(A) Trial 1



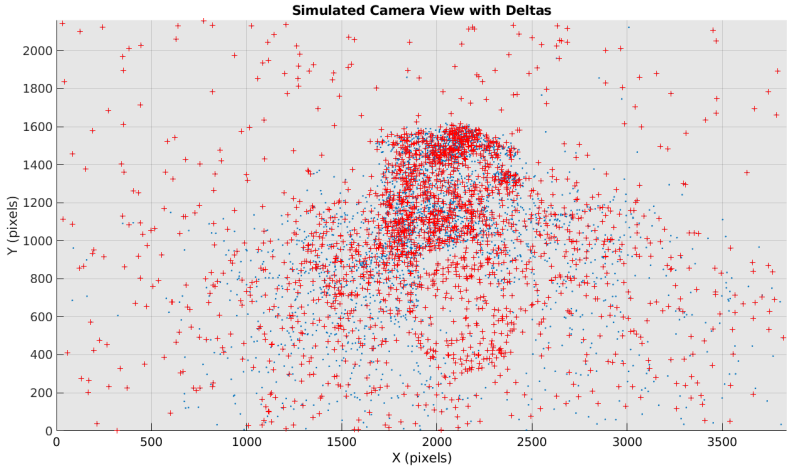
(B) Trial 2

FIGURE 4.11: Simulated Camera Views

Feature correlation with respect to the prior map projection in Fig. 4.5 is performed. The results of feature correlation for both trials are depicted in Fig. 4.12. For "Trial 2", very few feature points were successfully correlated with map points.



(A) Trial 1

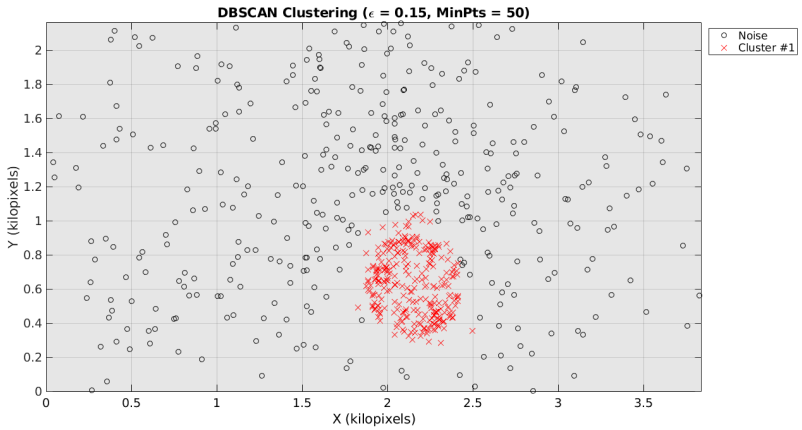


(B) Trial 2

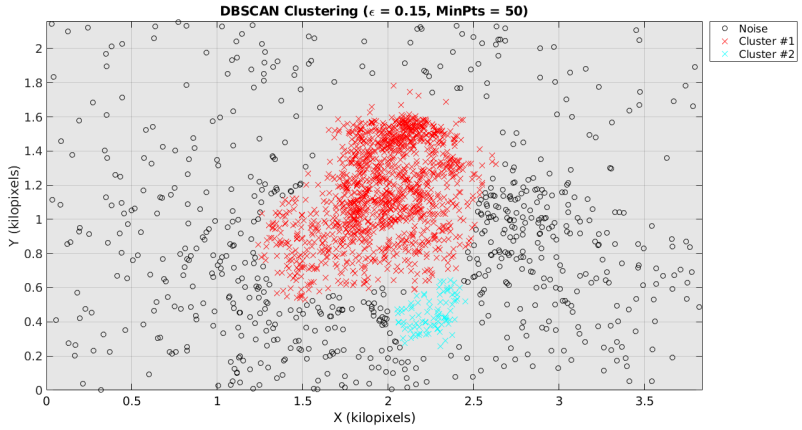
FIGURE 4.12: Simulated Camera Views with Deltas

The set of unmatched feature points for each trial is parsed with the DBSCAN clustering algorithm. The results are depicted in Fig. 4.13. The red points represent the cluster corresponding to the new object, while the black points correspond to noise. The degree to which this algorithm breaks down in the presence of significant noise is readily apparent. For "Trial 2", the algorithm incorrectly thinks that the chimney itself is a new object in the scene, while a small portion of the sphere is

correctly identified as a new object.



(A) Trial 1



(B) Trial 2

FIGURE 4.13: Clustering of Unmatched Feature Points

Chapter 5

Concluding Remarks

Through a fusion of real and simulated data, this work has demonstrated that the described method for determining and localizing changes in the real world with respect to a sparse prior map is feasible. A proof-of-concept algorithm has been implemented in C++, and a simulated environment has been developed in MATLAB. However, several limitations exist before a fully-functional real-time implementation on-board a MAV is realized. The SIFT feature algorithm does not have sufficient robustness against changing environmental conditions (such as lighting) for the proposed algorithm to function satisfactorily. Switching from SIFT to another feature algorithm is not feasible because COLMAP, upon which much of this work depends, employs SIFT internally, and outputs its reconstructed point clouds expressed in SIFT feature points. If the real-time requirement is relaxed, then employing textured dense reconstructions as prior maps as opposed to just sparse point clouds proves to be a promising avenue for future work. Change detection with fully colored images is expected to be far more robust than with feature point correlations. However, loading and rendering textured dense reconstructions is a computationally intensive task, one that is not currently feasible given the computational limitations on these MAVs.

Bibliography

[1] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, *et al.*, "Stanley: The robot that won the darpa grand challenge", *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[2] A. Broggi, M. Buzzoni, S. Debattisti, P. Grisleri, M. C. Laghi, P. Medici, and P. Versari, "Extensive tests of autonomous driving technologies", *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1403–1415, 2013.

[3] D. Scaramuzza, M. C. Achtelik, L. Doitsidis, F. Friedrich, E. Kosmatopoulos, A. Martinelli, M. W. Achtelik, M. Chli, S. Chatzichristofis, L. Kneip, *et al.*, "Vision-controlled micro flying robots: From system design to autonomous navigation and mapping in gps-denied environments", *IEEE Robotics & Automation Magazine*, vol. 21, no. 3, pp. 26–40, 2014.

[4] K. M. Pesyna Jr *et al.*, "Advanced techniques for centimeter-accurate gnss positioning on low-cost mobile platforms", PhD thesis, 2015.

[5] M. J. Murrian, C. W. Gonzalez, T. E. Humphreys, and T. D. Novlan, "A dense reference network for mass-market centimeter-accurate positioning", in *Position, Location and Navigation Symposium (PLANS), 2016 IEEE/ION*, IEEE, 2016, pp. 243–254.

[6] D. G. Lowe, "Object recognition from local scale-invariant features", in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, Ieee, vol. 2, 1999, pp. 1150–1157.

[7] P. Sturm, "Pinhole camera model", in *Computer Vision*, Springer, 2014, pp. 610–613.

[8] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors", *ArXiv preprint arXiv:1611.10012*, 2016.

[9] A. Fathi, Z. Wojna, V. Rathod, P. Wang, H. O. Song, S. Guadarrama, and K. P. Murphy, "Semantic instance segmentation via deep metric learning", *CoRR*, vol. abs/1703.10277, 2017. arXiv: 1703.10277. [Online]. Available: http://arxiv.org/abs/1703.10277.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications", *CoRR*, vol. abs/1704.04861, 2017. arXiv: 1704.04861. [Online]. Available: http://arxiv.org/abs/1704.04861.

[12] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision", *CoRR*, vol. abs/1512.00567, 2015. arXiv: 1512.00567. [Online]. Available: http://arxiv.org/abs/1512.00567.

[13] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited", in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[14] G. Bradski, "The opencv library", *Dr. Dobb's Journal of Software Tools*, 2000.

[15] G. Guennebaud, B. Jacob, *et al.*, *Eigen v3*, http://eigen.tuxfamily.org, 2010.

[16] A. Vedaldi and B. Fulkerson, "Vlfeat: An open and portable library of computer vision algorithms", in *Proceedings of the 18th ACM International Conference on Multimedia*, ser. MM '10, Firenze, Italy: ACM, 2010, pp. 1469–1472, ISBN: 978-1-60558-933-6. DOI: 10.1145/1873951.1874249. [Online]. Available: http://doi.acm.org/10.1145/1873951.1874249.

[17] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration", in *International Conference on Computer Vision Theory and Application VISSAPP'09)*, INSTICC Press, 2009, pp. 331–340.

[18] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.", in *Kdd*, vol. 96, 1996, pp. 226–231.

[19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.