3-22-2019

Monocular Visual Odometry for Fixed-Wing Small Unmanned Aircraft Systems

Kyung M. Kim

Monocular Visual Odometry for Fixed-Wing Small Unmanned Aircraft Systems

THESIS

Kyung M. Kim, Capt, USAF

AFIT-ENG-MS-19-M-036

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-19-M-036

MONOCULAR VISUAL ODOMETRY FOR FIXED-WING SMALL UNMANNED AIRCRAFT SYSTEMS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Kyung M. Kim, BS

Capt, USAF

March 2019

AFIT-ENG-MS-19-M-036

MONOCULAR VISUAL ODOMETRY FOR FIXED-WING SMALL UNMANNED AIRCRAFT SYSTEMS

THESIS

Kyung M. Kim, BS
Capt, USAF

Committee Membership:

Dr. R. C. Leishman
Chair

Dr. J. F. Raquet
Member

Dr. S. L. Nykl
Member

Abstract

The popularity of small unmanned aircraft systems (SUAS) has exploded in recent years and seen increasing use in both commercial and military sectors. A key interest area for the military is to develop autonomous capabilities for these systems, of which navigation is a fundamental problem. Current navigation solutions suffer from a heavy reliance on a Global Positioning System (GPS). This dependency presents a significant limitation for military applications since many operations are conducted in environments where GPS signals are degraded or actively denied. Therefore, alternative navigation solutions without GPS must be developed and visual methods are one of the most promising approaches. A current visual navigation limitation is that much of the research has focused on developing and applying these algorithms on ground-based vehicles, small hand-held devices or multi-rotor SUAS. However, the Air Force has a need for fixed-wing SUAS to conduct extended operations.

This research evaluates current state-of-the-art, open-source monocular visual odometry (VO) algorithms applied on fixed-wing SUAS flying at high altitudes under fast translation and rotation speeds. The algorithms tested are Semi-Direct VO (SVO), Direct Sparse Odometry (DSO), and ORB-SLAM2 (with loop closures disabled). Each algorithm is evaluated on a fixed-wing SUAS in simulation and real-world flight tests over Camp Atterbury, Indiana. Through these tests, ORB-SLAM2 is found to be the most robust and flexible algorithm under a variety of test conditions. However, all algorithms experience great difficulty maintaining localization in the collected real-world datasets, showing the limitations of using visual methods as the sole solution. Further study and development is required to fuse VO products with additional measurements to form a complete autonomous navigation solution.

Acknowledgements

I would like to thank my adviser, committee members, and ANT Center staff for all of their help and support throughout this work.

Kyung M. Kim

Table of Contents

Page

Abstract iv

Acknowledgements v

List of Figures viii

List of Tables xi

I. Introduction 1

1.1 Problem Statement 1
1.2 Research Goals 2
1.3 Hypothesis 2
1.4 Approach 2
1.5 Assumptions/Limitations 3
1.6 Contributions 4
1.7 Thesis Overview 4

II. Literature Review 5

2.1 Autonomous SUAS Navigation 5
2.2 VO versus VSLAM 6
2.3 Notation 7
2.4 Camera Calibration 7
2.5 Monocular VO 10
2.5.1 Feature-based Methods 11
2.5.2 Direct Methods 16
2.6 Related Works 17
2.7 Summary 29

III. Methodology 31

3.1 Dependencies 31
3.1.1 Middleware 31
3.1.2 AftrBurner Engine 32
3.1.3 Qt 34
3.1.4 OpenCV 34
3.1.5 ArduPilot 34
3.1.6 Vimba SmartCables Driver 35
3.1.7 Dependency Conflict 35
3.2 System Design 36
3.2.1 VO Nodes 36
3.2.2 Camera Calibration 37

Page

3.2.3 Autonomy Simulator . . . 39
3.2.4 Trajectory Viewer . . . 45
3.2.5 Real-World Aircraft . . . 46
3.3 Experimental Design . . . 47
3.3.1 Assumptions/Limitations . . . 47
3.3.2 Simulation . . . 48
3.3.3 Real-World . . . 50
3.3.4 VO Algorithms . . . 52
3.3.5 Processing Platform . . . 52
3.3.6 Performance Metrics . . . 53
3.3.7 Summary . . . 53

IV. Results & Analysis . . . 54

4.1 Simulation Results . . . 54
4.1.1 Default Parameters at 10 m/s . . . 54
4.1.2 Default Parameters at 25 m/s . . . 58
4.1.3 Framerate at 5 fps . . . 59
4.1.4 Lowered VO thresholds . . . 61
4.1.5 SVO Keyframe Selection Distance . . . 64
4.2 Real-World Flight Test Results . . . 65
4.3 Analysis . . . 72
4.4 Summary . . . 73

V. Conclusion . . . 74

5.1 Experiment . . . 74
5.2 Future Work . . . 74
Bibliography . . . 76

List of Figures

Figure		Page
1.	Pinhole Model	8
2.	Lens Distortion	9
3.	Calibration Checkerboard	10
4.	Pixel Density. Green pixels shows the pixels used for VO calculation.	11
5.	ORB Features	12
6.	Basic Components of a Feature-based VO System	13
7.	Epipolar Constraint Components	13
8.	Basic Components of Direct Methods	16
9.	LSD-SLAM	20
10.	ORB-SLAM	23
11.	SVO	24
12.	Carson VO Variants	29
13.	Virtual Grand Canyon	33
14.	VO ROS/LCM Node	37
15.	Camera Calibration Modules	38
16.	Virtual Monocular Calibration Module	38
17.	Camera Calibration GUI	39
18.	Autonomy Simulator Overview	40
19.	Simulated Flight Trajectory	42
20.	Simulated Flight with DSO	43
21.	Simulated Flight with ORB-SLAM2	44
22.	Simulated Flight with SVO	44

Figure Page

23. Trajectory Viewer 45

24. Trajectory Plot Viewer 46

25. Simulation Test Trajectory 49

26. Box Flight Pattern 50

27. Cloverleaf Flight Pattern 51

28. Grid Flight Pattern 51

29. Simulation VO Performance with Default Parameters at 10 m/s 55

30. Simulation Trajectories at 10 m/s, 25 deg/s, 300 m 57

31. Simulation Pose Data at 10 m/s, 25 deg/s, 300 m 58

32. Simulation VO Performance with Default Parameters at 25 m/s 59

33. Simulation VO Performance at 5 fps 61

34. Simulation DSO Image Gradient Threshold Performance 62

35. Simulation ORB-SLAM2 FAST Threshold Performance 63

36. Simulation SVO FAST Threshold Performance 63

37. Simulation SVO Keyframe Distance Performance 64

38. Simulation SVO Trajectories at 25 m/s, 25 deg/s, 350 m 65

39. Saturated Low Texture Image 66

40. Real-World Truth Trajectory 67

41. Real-World ORB-SLAM2 Trajectory 68

42. Real-World ORB-SLAM2 Pose 68

43. Real-World ORB-SLAM2 Position Error 69

44. ORB-SLAM2 Trajectory from Real-World Trajectory and Virtual Imagery 70

Figure Page

45. ORB-SLAM2 Pose Data from Real-World Trajectory and Virtual Imagery 70

46. Real-World SVO Trajectory 71

List of Tables

Table Page

1. Simulation VO Accuracy with Default Parameters at 10 m/s 56

2. Simulation VO Accuracy with Default Parameters at 25 m/s 60

MONOCULAR VISUAL ODOMETRY FOR FIXED-WING
SMALL UNMANNED AIRCRAFT SYSTEMS

I. Introduction

In 2016, the United States Air Force released the SUAS Flight Plan highlighting the importance of SUAS in modern military operations and the need to further develop and integrate these systems into the Air Force weapons set [1]. The Flight Plan also addresses the need for higher degrees of autonomy to alleviate operational requirements, protect against lost communication links, and enable integration with other manned and unmanned aircraft. A central problem to solve in order to enable autonomous operations is navigation.

1.1 Problem Statement

Current SUAS navigation solutions rely heavily on GPS which can be degraded or denied during missions. Monocular Visual Odometry (VO) is an alternative navigation solution that has made significant progress in the last decade, only recently producing viable solutions that can be run on small mobile platforms with limited resources. Monocular VO uses the information from images produced by a single camera to estimate the camera's motion [2]. Monocular VO is attractive since cameras incur a low energy cost, images provide a wealth of information, and Air Force SUAS are already equipped with cameras. However, the majority of current monocular VO research has focused on ground-based or quadrotor platforms. There have not been many studies deploying these algorithms on fixed-wing SUAS which typically fly at higher altitudes and speeds and undergo different types of motions. This

thesis attempts to answer the following problem statements: are current state-of-the-art monocular VO algorithms viable as a front-end solution for fixed-wing SUAS navigation and what are the limitations under high-speed maneuvers?

1.2 Research Goals

This research evaluates the real-time performance of current state-of-the-art monocular VO algorithms on a fixed-wing SUAS under high-speed maneuvers. Three algorithms are applied over multiple trajectories in both simulation and real-world flight tests. The accuracy and robustness of each algorithm is analyzed and compared under different flight conditions.

1.3 Hypothesis

This work hypothesizes that current state-of-the-art monocular VO algorithms are viable starting points for an alternative navigation solution for fixed-wing SUAS. However, a complete and robust solution will require fusing VO outputs with additional measurements from sensors such as an IMU and a backend system executing loop closures.

1.4 Approach

The performance of three VO algorithms are tested and analyzed in simulation and real-world flight tests: SVO [3], DSO [4], and ORB-SLAM2 [5] with loop closures disabled. In order to conduct the simulations, a high-fidelity virtual environment is created that allows the algorithms to be tested on a variety of trajectories and terrains. The terrains consist of scaled satellite images that are mapped to elevation data and GPS coordinates to accurately reflect real-world outdoor settings. A virtual

camera streams live imagery of this terrain while flying above it and the imagery stream is used as input for the algorithms.

The virtual environment [6] interfaces with an open-source autopilot software that provides an interchangeable connection to a virtual autopilot or actual hardware. This allows the generation of simulated trajectories with realistic flight dynamics as well as the ability to project the status of a real SUAS. The virtual environment enables recording and playback of truth data with multiple VO state estimation data.

Real-world flight tests are conducted over Camp Atterbury, Indiana where flight data is gathered for real-time playback and processing post-flight. Both the flight data and VO outputs can be projected into the virtual environment to analyze the performances in real-time with intuitive visualization of scale and 6-DOF pose.

1.5 Assumptions/Limitations

The following assumptions and limitations are made during this experiment:

1. The camera faces downward in the aircraft's body frame.

2. The virtual terrain image resolution is representative of real-world terrain when viewed from mission altitudes.

3. There is negligible error from the lack of parallax from non-elevation features in the virtual terrain such as trees when viewed from mission altitudes.

4. There is negligible distortion in the virtual terrain image from mapping the 2D image onto the 3D elevation map for Camp Atterbury.

5. VO algorithms are initialized when the SUAS is flying level and the camera is facing down towards the Earth's center. The altitude at initialization is equal to the absolute scale of the estimated trajectory.

6. A photometric camera calibration is not provided.

1.6 Contributions

This thesis makes the following contributions to the field of VO:

1. **VO/VSLAM virtual environment.** This work presents a virtual environment ideal for testing and comparing VO and VSLAM algorithms in outdoor environments. Realistic trajectories over real-world terrain can be generated and perfect truth data is readily available. This facilitates easy testing under a wide variety of flight conditions and SUAS parameters. Real-time reprojections of VO results scaled to the world environment give an intuitive understanding of algorithm performances during flight. Playback and overlays of multiple VO estimated trajectories allow easy comparison and analysis. Although the algorithms tested in this research are limited to VO, this virtual environment is easily extensible to SLAM algorithms.

2. **Fixed-wing VO.** This work analyzes the performance of current state-of-the-art monocular VO algorithms on a fixed-wing SUAS platform. It proposes a VO algorithm for use as a front-end in a larger SLAM framework incorporating multiple sensors and loop closure capabilities.

1.7 Thesis Overview

This thesis is organized into five chapters. Chapter 2 gives a background and presents related research on monocular vision navigation. Chapter 3 describes the test systems and experiment methodology. Chapter 4 provides the test results and analysis. Chapter 5 summarizes the research and provides recommendations for future work.

II. Literature Review

This chapter provides a background on autonomous navigation and discusses the application of visual solutions on a SUAS. It introduces mathematical notations for transformations and calculations on images. It presents core concepts behind camera calibration and VO. This chapter ends by surveying related works that have contributed to the field.

2.1 Autonomous SUAS Navigation

The fundamental components used for onboard autonomous SUAS navigation are the Inertial Measurement Unit (IMU) and GPS. The IMU contains accelerometers and gyroscopes to calculate linear and angular movements but accumulates error from drift over time and requires a GPS to continually correct this drift. However, GPS signals can be degraded, lost or actively denied leading to erroneous measurement inputs being provided to onboard control algorithms. Therefore, alternative methods are required to either supplement or replace GPS as a navigation measurement source [7].

Using a monocular camera on a SUAS for navigation is appealing due to the relatively light weight, low monetary and energy cost, the discreet nature of passive sensing, and the wealth of information available in an image. This can be compared to other sensors such as lasers which have met with great success on larger ground-based platforms but are typically too big and energy-intensive to use on smaller aerial platforms with restrictive size and power limitations [8] [9]. Another navigation method involves the use of stereo cameras which has the distinct advantage over a monocular scheme in that the absolute scale of the environment and position can be directly computed using the stereo baseline (distance between the two cameras) [10]

[11]. However, this degrades to the monocular scheme when the distance from the cameras to the scene is much greater than the baseline [2]. This is typically the case for fixed-wing SUAS where the vehicle flies at high altitudes. RGB-D sensors have also been used to directly obtain absolute scale but the depth sensors are limited in range and not viable at high altitudes [12] [13] [14].

2.2 VO versus VSLAM

Research in visual navigation primarily falls under two categories: VO and VSLAM. VO incrementally computes the relative motion of a camera and tracks a more localized environment [2]. VSLAM falls under the larger umbrella of SLAM which aims to build and maintain a global map of the environment in order to calculate a camera's position and maintain a globally consistent trajectory [7].

Being an odometry method, VO does not maintain a global map to calculate its state and therefore suffers from drift similar to an IMU [2]. Therefore, it also requires additional mechanisms to periodically constrain its error. These measures can take the form of a GPS or the incorporation of loop closures in a SLAM system.

As the name suggests, VSLAM is a SLAM solution in which visual sensors are used as the primary input. A SLAM system is usually divided into front-end and back-end systems [15]. The front-end system provides an open loop odometry implementation and in VSLAM this is usually through VO. The back-end builds the globally consistent map, fuses additional inputs if available, localizes the robot relative to observed landmarks, and calculates loop closures to correct the trajectory if a previously observed landmark is re-encountered [16].

2.3 Notation

This section defines the notation that will be used to describe core concepts and equations for VO. The rotation and translation of a body's coordinate frame from a timestep or frame $k-1$ to k are described by the transformation matrix $T_{k-1}^{k} \in SE(3)$:

$$T_{k-1}^{k} = \begin{bmatrix} R_{k-1}^{k} & t_{k-1}^{k} \\ 0 & 1 \end{bmatrix} \tag{1}$$

where $R_{k-1}^{k} \in SO(3)$ is the rotation matrix that expresses points from frame $k-1$ in frame k and t_{k-1}^{k} describes the translation vector of points from frame k to $k-1$ expressed in frame k.

The image taken at timestep k is given by I_k. A pixel coordinate for I_k is given by $\mathbf{u} = (u, v)^{\mathrm{T}}$. The pixel's corresponding 3D coordinate relative to the camera's frame at timestep k is denoted by $\mathbf{p}^k = (x, y, z)^{\mathrm{T}}$.

2.4 Camera Calibration

Camera calibration is required in order to calculate a camera's intrinsic parameters and model the distortion effects of the lens [17]. The intrinsic parameters define the mapping between a 3D point and its 2D image projection. The pinhole camera model is one of the most common models used for calibration and is the model used to test the various VO algorithms in this work. The pinhole model assumes that all the light rays coming into the camera pass through one point in the center of the lens as shown in Figure 1.

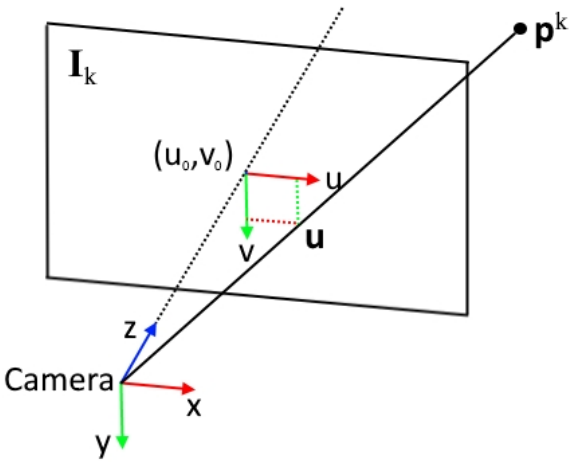

Figure 1. Pinhole Model

With this model, the projection of point $\mathbf{p}^k$ onto the image plane $\mathbf{I}_k$ becomes:

$$\begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z} K \mathbf{p}^k = \frac{1}{z} \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2}$$

where K is the camera matrix containing the intrinsic parameters f_x, f_y, u_0, and v_0. f_x and f_y are the focal lengths in the respective directions and $(u_0, v_0)^{\mathrm{T}}$ is the coordinate of the projection center. $K\mathbf{p}^k$ is scaled down by z in order to normalize the 3D coordinates onto the same image plane [17].

Additionally, camera lenses cause radial and tangential distortions in an image so that straight lines bulge and curve away from the center as shown in Figure 2. Given $x' = x/z$, $y' = y/z$, and $r = \sqrt{x'^2 + y'^2}$, the radial distortion is modeled by Equation 3, the tangential distortion by Equation 4, and the total distortion by Equation 5. $(d_0, d_1, d_2, d_3, d_4)$ are the distortion coefficients [18].

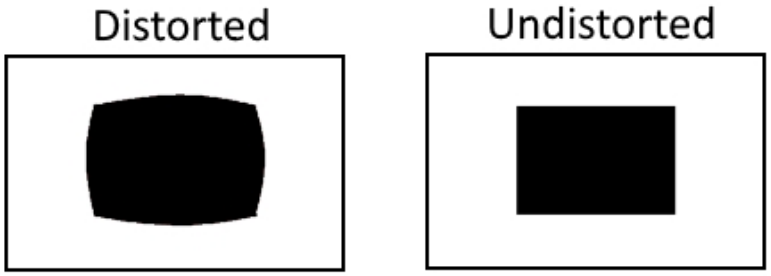

Figure 2. Lens Distortion

$$\begin{bmatrix} x'_{radial\ distortion} \\ y'_{radial\ distortion} \end{bmatrix} = (1 + d_0 r^2 + d_1 r^4 + d_2 r^6) \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} x'_{tangential\ distortion} \\ y'_{tangential\ distortion} \end{bmatrix} = \begin{bmatrix} 2d_3 x' y' + d_4(r^2 + 2x'^2) \\ d_3(r^2 + 2y'^2) + 2d_4 x' y' \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} x'_{distortion} \\ y'_{distortion} \end{bmatrix} = \begin{bmatrix} x'_{radial\ distortion} \\ y'_{radial\ distortion} \end{bmatrix} + \begin{bmatrix} x'_{tangential\ distortion} \\ y'_{tangential\ distortion} \end{bmatrix} \tag{5}$$

One of the most common methods of performing a camera calibration is by taking pictures of a flat checkerboard of known size at multiple orientations and positions as shown in Figure 3 [18]. The intrinsic parameters in K and the distortion coefficients $(d_0, d_1, d_2, d_3, d_4)$ can then be backed out and used to undistort images taken with the camera as well as calculate 3D position and orientation changes in VO algorithms.

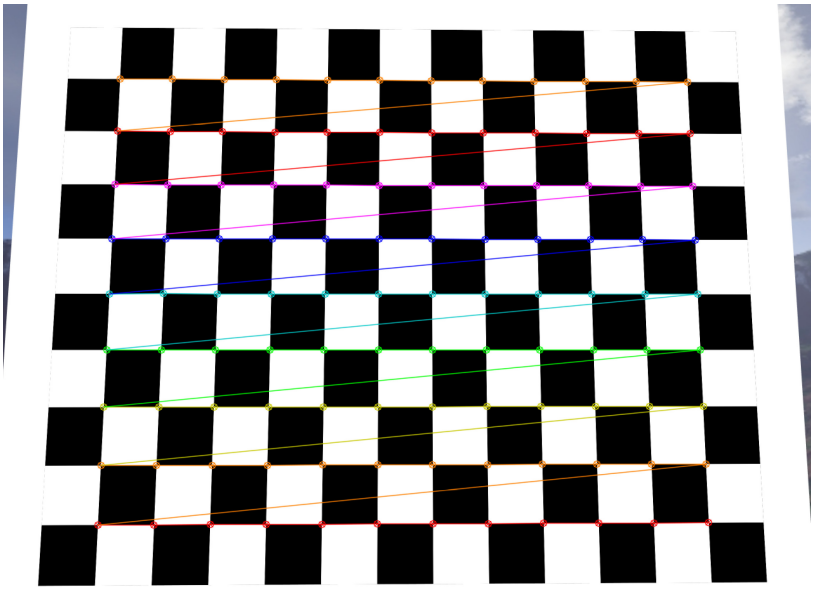

Figure 3. Calibration Checkerboard

2.5 Monocular VO

A core concept of monocular VO is to incrementally calculate the relative transformations of the camera T_{k-1}^{k} using the image data from the images I_{k-1} and I_k. The camera's pose relative to the local starting frame at any point in the trajectory can be recovered by chaining together all of the preceding incremental transformations $T_1^0 \times T_2^1 \times ... \times T_{n-1}^{n-2} \times T_n^{n-1}$ [19] [20]. Many VO algorithms periodically designate certain frames with unique image features as a keyframe r_n and only hold onto a set amount of keyframes [4] [5] [20] [21] [22] [23]. The keyframes keep track of the image features and their estimated 3D positions, forming a local map. The transformations for subsequent regular frames are then calculated relative to the latest keyframe $T_k^{r_n}$ and new measurements are used to refine the 3D positions of observed map points in the previous keyframes until a new keyframe is designated.

VO algorithms differ by the density of pixels used in an image and by the method of image comparison used. Pixel density falls into three categories: sparse, dense,

and semi-dense [5] [22] [24]. The two predominant image comparison methods are feature-based and direct methods [4] [5].

Sparse VO algorithms select a small number of pixels at key points throughout an image. Dense methods on the other hand use every pixel in the image. Semi-dense methods use pixels from regions of an image. Semi-dense image regions can be selected based on their image intensity gradients [24]. An example of the different pixel densities used from an image is shown in Figure 4. The green pixels in each image show the pixels used for the VO algorithm.

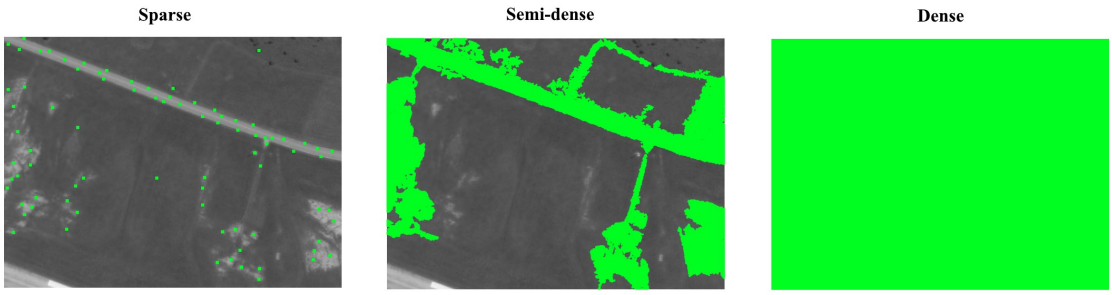

Figure 4. Pixel Density. Green pixels shows the pixels used for VO calculation.

2.5.1 Feature-based Methods.

Feature-based methods identify local keypoints or areas of interest throughout an image and extract feature vector representations, or descriptors, for each point which are then matched between sets of images [25]. The relative transformation between the corresponding camera poses for the image frames is calculated to optimize the geometric error between the feature positions. Features remain invariant to transformations and changes in lighting conditions to calculate and match the same features across different images. Extracting features to meet this criteria and matching them between images can be costly operations. Some of the most prevalent features used are SIFT [26], SURF [27], FAST [28], and ORB [29]. Figure 5 shows an example of ORB features detected and matched across two images of the same scene taken at

different positions.

Figure 5. ORB Features

The fundamental components of a feature-based VO algorithm is shown in Figure 6. Features are first extracted from the latest image and matched or tracked across previous images. The transformation describing the motion from the previous frame to the current frame is then calculated using the geometry of the matched feature points between the different images. This commonly involves utilizing the epipolar constraint shown in Figure 7 [17]. A feature viewed by the same camera in two images I_{k-1} and I_k has a 3D world coordinate $\mathbf{p}^w$ that forms a plane with the camera centers at frames $k-1$ and k. This plane is the epipolar plane. Epipolar lines are formed by the intersections of the epipolar plane with the image planes I_{k-1} and I_k. The projections of $\mathbf{p}^w$ onto I_{k-1} and I_k are at the 3D coordinates $\mathbf{p}'^{k-1}$ and $\mathbf{p}'^{k}$. These points lie on their respective epipolar lines and are described relative to the camera's coordinate frames at $k-1$ and k.

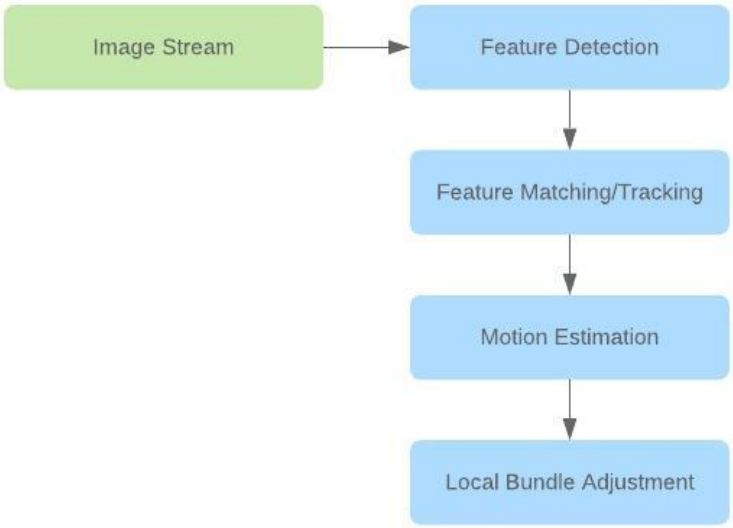

Figure 6. Basic Components of a Feature-based VO System

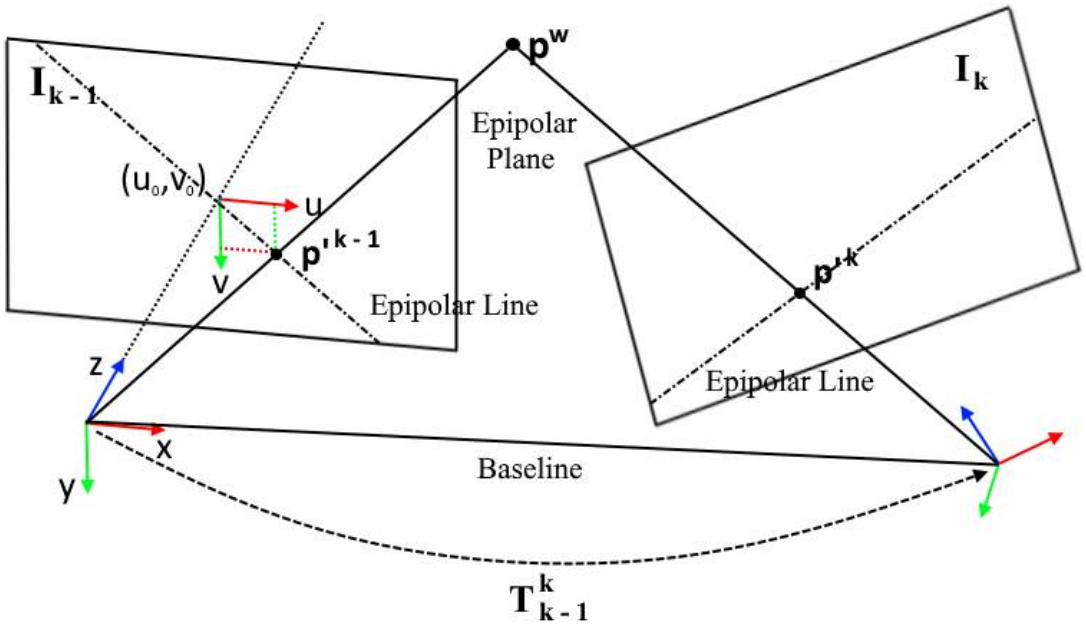

Figure 7. Epipolar Constraint Components. The epipolar plane is formed by the camera centers and feature point $\mathbf{p}^w$. Epipolar lines are formed by the intersection of the epiploar plane and image planes

If $\mathbf{T}^k_{k-1}$ is the transform that rotates and translates the camera frame from $k-1$

to k, the epipolar constraint relies on the fact that the vectors t_{k-1}^{k} (from Equation 1), $\mathbf{p'}^{k-1}$, and $\mathbf{p'}^{k}$ are all coplanar on the epipolar plane. Expressing these vectors relative to the camera at frame k would mean that the vectors t_{k-1}^{k}, $R_{k-1}^{k}\mathbf{p'}^{k-1}$, and $\mathbf{p'}^{k}$ are all coplanar. This property is mathematically represented by Equation 6 [17]. Conceptually, the cross product of t_{k-1}^{k} and $R_{k-1}^{k}\mathbf{p'}^{k-1}$ yields a vector that is perpendicular to the epipolar plane formed by the two vectors. This cross product is consequently perpendicular to $\mathbf{p'}^{k}$. Therefore, the dot product between the perpendicular vector and $\mathbf{p'}^{k}$ must be zero. This equation is normally rewritten in the form of Equation 7 where E is known as the essential matrix and is equal to $t_{k-1}^{k} \times R_{k-1}^{k}$. This equation can be extended to incorporate the feature point's corresponding 2D image coordinates $\mathbf{u}^{k-1}$ and $\mathbf{u}^{k}$. For the pinhole model, this would involve the camera matrix K as shown in Equation 8. Rearranging Equation 8 gives us Equation 9 which is normally rewritten as Equation 10 where F is known as the fundamental matrix and is equal to $(K^{-1})^{\mathrm{T}}EK^{-1}$ [17].

$$\mathbf{p'}^{k\mathrm{T}}(t_{k-1}^{k} \times R_{k-1}^{k}\mathbf{p'}^{k-1}) = 0 \tag{6}$$

$$\mathbf{p'}^{k\mathrm{T}}E\mathbf{p'}^{k-1} = 0 \tag{7}$$

$$(K^{-1}\mathbf{u}^{k})^{\mathrm{T}}E(K^{-1}\mathbf{u}^{k-1}) = 0 \tag{8}$$

$$\mathbf{u}^{k\mathrm{T}}(K^{-1})^{\mathrm{T}}EK^{-1}\mathbf{u}^{k-1} = 0 \tag{9}$$

$$\mathbf{u}^{k\mathrm{T}}F\mathbf{u}^{k-1} = 0 \tag{10}$$

Given a set of matched image points corresponding to a feature, the essential matrix is calculated in order to extract the rotation and translation of the camera between the two frames. Random Sample Consensus (RANSAC) is also a standard method used for outlier rejection [30]. RANSAC calculates motion model estimates from a random sample of matched feature points and verifies it against other corresponding feature samples. The model that best fits the data points is selected and this process is repeated for a desired number of iterations.

An issue with monocular VO algorithms is the inability to compute the absolute scale of the translation since the absolute depth information is lost during the projection of 3D points onto 2D images. The relative scale of transformations can be computed and propagated across frames although the error in scale would grow unconstrained without continuous corrections. Therefore, monocular VO must be supplemented by fusing measurements from other devices such as an IMU in order to extract the absolute scale [31] [32] [33]. However, this visual-inertial odometry configuration would still require additional corrections from loop closures or GPS [2].

After computing the rotation and translation, this estimate can be refined through local bundle adjustment. Bundle adjustment involves tracking feature points and optimizing the estimate over a sliding window of the last n image frames [34] [35]. This is accomplished by solving for the camera pose that minimizes the reprojection error of the tracked features into the image at the current frame k as shown in Equation 11. The reprojection error can be solved using a nonlinear least-squares algorithm such as the Gauss-Newton or Levenberg-Marquardt method [36]:

$$T_w^k = \arg_T \min \sum_i ||\mathbf{u}_i - \pi(T\mathbf{p}_i^w)||^2 \tag{11}$$

where π describes the mapping of 3D coordinates to the 2D points on the image plane.

2.5.2 Direct Methods.

Direct methods operate directly on the pixel intensity values in the images. The relative pose transformations are calculated by finding the optimal transformation that minimizes the photometric error of the selected pixels between the different images. This tends to be faster than feature-based methods since it avoids the costly computation steps of extracting features and matching them between images as shown in Figure 8. However, changes in lighting conditions are a major source of error [37].

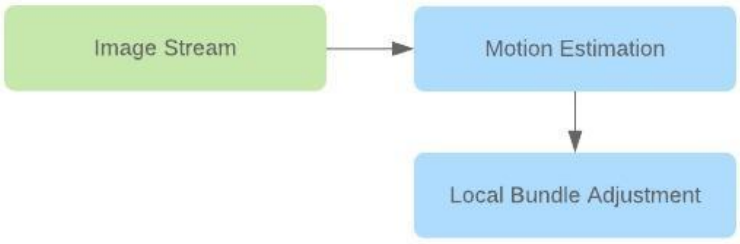

Figure 8. Basic Components of Direct Methods

The transformation between frames that will minimize the photometric error can be found through Equation 12. In this equation, image points from I_{k-1} are projected into I_k using their corresponding depth values $d_{\mathbf{u}_i}$ and the transformation estimate T_{k-1}^k. The intensity values at the image coordinates in both I_{k-1} and I_k are then compared. Similar to the bundle adjustment reprojection error, the photometric error is also nonlinear and a solution can be found by using a nonlinear least-squares algorithm. Bundle adjustment can then be used to optimize the pose over the last n frames similar to feature-based methods [3].

$$T_{k-1}^k = \arg_T \min \sum_i ||I_k(\pi(T\pi^{-1}(\mathbf{u}_i^{k-1}, d_{\mathbf{u}_i}))) - I_{k-1}(\mathbf{u}_i^{k-1})||^2 \tag{12}$$

Until recently, feature-based methods were the predominant approach with many algorithms using sparse pixels, especially in the earliest solutions. However, direct

methods have been gaining in popularity along with denser pixel schemes. Currently, real-time VO implementations for SUAS and other small mobile platforms are limited to semi-dense schemes due to the constrained computing environment.

2.6 Related Works

The first work using visual input to calculate a robot's motion was accomplished by Moravec in the 1980s [38]. In this work, a sliding monocular camera was used on a planetary rover in a method that essentially boils down to a stereo scheme. The rover intermittently makes stops during a trajectory at which the camera slides along a rail and takes nine pictures equal distances apart. Corner features are extracted, matched and triangulated to determine the 3D point locations. The feature locations are then used to estimate the robot's motion. However, it wasn't until 2004 that Nister et al. implemented the first real-time large-scale VO solution [39]. This VO algorithm uses Harris corners [40] for feature detection and RANSAC for robust motion estimation. It was tested on an autonomous ground vehicle and was able to process frames at 13 Hz. Nister et al. also first introduced the term VO in this paper. The first real-time VSLAM solution was produced by Davison et al. in 2007 and was called MonoSLAM [41]. MonoSLAM maintains a probabilistic 3D map of 100 corner features detected using the Shi and Tomasi operator [42]. This map is then used to localize the camera at 30 Hz. It was demonstrated on a humanoid robot walking around in an indoor environment.

Also in 2007, Klein and Murray developed Parallel Tracking and Mapping (PTAM) which introduced the idea of splitting tracking and mapping into separate threads to achieve higher overall performance and accuracy [23]. By not tying the tracking process to the map update, the tracking thread is able to execute more detailed image processing on each frame and only supply keyframes containing non-redundant

features to the mapping thread. The mapping thread is also able to employ more expensive but accurate methods of updating its map such as bundle adjustment due to the relaxed time constraint in not having to incrementally update the map on every frame. The tracking thread can then provide real-time pose estimates of the camera using the currently built map. PTAM was originally designed for augmented reality applications in small indoor desktop environments and its mapping system is not scalable for large scenes. However, its scheme of separating tracking and mapping into separate threads remains the foundational architecture for modern VO and VSLAM algorithms.

In 2011, Newcombe et al. introduced Dense Tracking and Mapping (DTAM), a fully dense and direct algorithm designed to be highly parallelizable and employed on a Graphics Processing Unit (GPU) [22]. By combining computer vision and graphics techniques, DTAM builds dense 3D surface models of the environment, textures them with images, and reprojects them into a virtual camera. Direct methods are used to align the whole image and find the pose that minimizes the reprojection error between the virtual and live images. Pixels whose photometric error falls above a threshold are ignored to allow continued tracking even with the introduction of new unmodelled objects in the scene. DTAM is significantly more robust under occlusions, camera blur and defocus than other feature-based methods while being able to maintain a framerate of 30 Hz. However, most VO and VSLAM solutions for SUAS and other mobile robotics currently focus on implementing solely CPU-based algorithms which limits the ability to process every pixel per image. DTAM is not publicly available.

Engel et al. produced a set of real-time direct VO and VSLAM algorithms for monocular cameras starting with Semi-Dense VO in 2013 [43]. This algorithm maintains a semi-dense inverse depth map for image regions with non-negligible gradients. Tracking is conducted by minimizing the photometric error using the iterative Gauss-

Newton method over four coarse-to-fine pyramid levels. A feature-based method is required for the first two frames in order to compute the first transformation and initialize the inverse depth map. This algorithm is able to maintain a tracking rate of 30 Hz and a mapping rate of 15 Hz.

Engel et al. built upon the concepts of Semi-Dense VO to create Large-Scale Direct SLAM (LSD-SLAM) in 2014 [24]. Figure 9 shows an overview of the algorithm. LSD-SLAM is made up of three parts: tracking, depth map estimation, and map optimization. LSD-SLAM stores the inverse depth map over keyframes and tracks the camera pose relative to the latest keyframe in the map as in Semi-Dense VO. Subsequent regular frames are used to refine the depth map of the latest keyframe. A new keyframe is added if the camera is too far from this keyframe. The keyframe is then replaced and added to the global map for map optimization. The map optimization component utilizes OpenFABMAP, an open-source Fast Appearance-Based Mapping algorithm, to detect large loop closures [44]. OpenFABMAP uses a bag-of-words approach which represents images through feature detectors, or visual words, and measures the similarity between images by the visual word histograms. This allows the algorithm to efficiently find similar images out of large datasets in order to detect loops. LSD-SLAM continuously optimizes the map by representing the keyframes and their relative transformations as a pose-graph and employing $\mathbf{g^2o}$, an open-source nonlinear graph optimization algorithm [45].

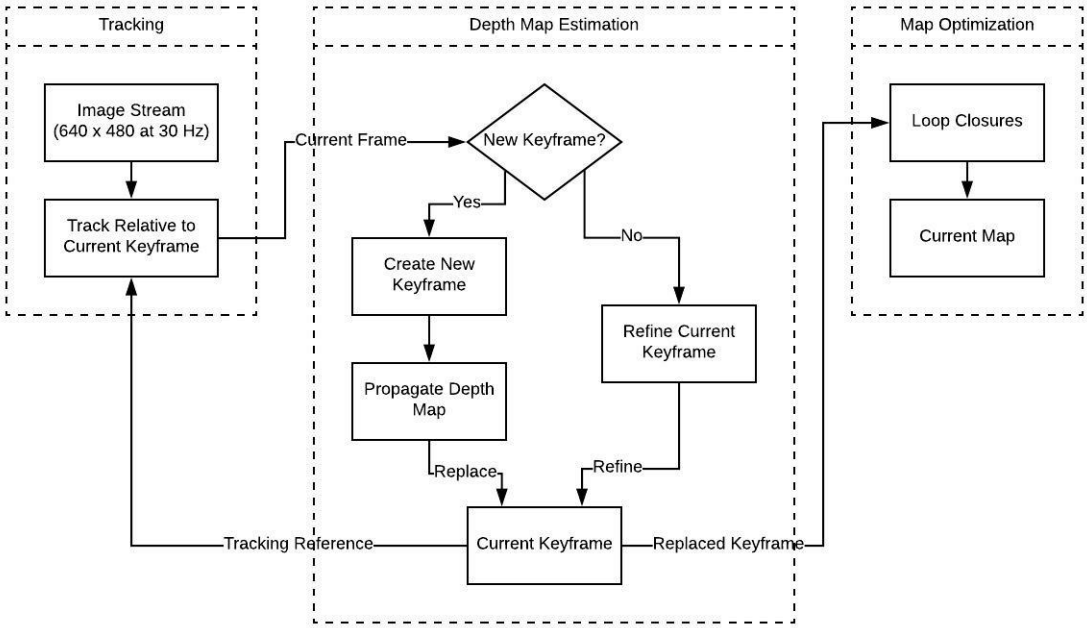

Figure 9. LSD-SLAM

DSO was released by Engel et al. in 2016 [4]. This algorithm not only utilizes geometric camera calibration to model the camera's intrinsic parameters, but it also incorporates an optional photometric camera calibration to model the camera's exposure response and pixel attenuation (vignetting) [46]. Photometric camera calibration is not necessary in feature-based methods since features are selected to be invariant to changes in lighting conditions but is helpful in direct methods since they depend solely on pixel intensities. In DSO, new frames are tracked relative to the latest keyframe. If the frame is not designated as a new keyframe, it is discarded. If a new keyframe is created, the photometric error is optimized over a sliding window of the latest seven keyframes. DSO tracks a fixed, sparse set of pixels across all keyframes. Candidate pixels in an image are selected for tracking by dividing the image into blocks of size $d \times d$ and calculating a region-adaptive gradient threshold for each block based on the median gradient of the pixels in the block. The pixel with the highest gradient greater

than the threshold is chosen. In order to be able to include pixels in regions with small gradients, the process is repeated again over larger block sizes of $2d$ and $4d$ with weaker gradient thresholds. LSD-SLAM and DSO are both currently open-source.

Several modifications and extensions have been made by members of the Computer Vision Group at the Technical University of Munich following on the work on DSO. Von Stumberg et al. created Visual Inertial DSO (VI-DSO) which tightly integrates IMU information with DSO and jointly optimizes the photometric and IMU measurement errors [47]. This allows for the calculation of scale and increased robustness during fast maneuvers or low-textured areas. Gao et al. adapted DSO into a VSLAM system in LDSO [48]. LDSO detects loop closures through the bag-of-words method which requires the use of repeatable features. In order to minimize overhead, LDSO favors corner features and uses them for both camera tracking as well as loop closures. Currently, only LDSO is available as open-source.

ORB-SLAM is one of the leading feature-based VSLAM algorithms and was created by Mur-Artal et al. in 2015 [49]. The system overview is shown in Figure 10. ORB-SLAM involves three threads for tracking, mapping, and loop closing. ORB-SLAM maintains an undirected weighted graph of keyframes that are linked to each other based on the number of shared map points in the keyframe images. This covisibility graph is used for loop closures and pose graph optimizations. However, in order to increase efficiency, a subset of the covisibility graph is also maintained as a spanning tree and is called the essential graph. The essential graph contains all keyframe nodes but only connects keyframes sharing the most map points.

ORB-SLAM extracts ORB features from an image and uses them for tracking, relocalization, and loop detection which allows for high efficiency. In the tracking thread, an initial pose estimation is obtained for the current frame by either using a constant velocity motion model from the last frame if tracking was successful in the

last frame or, if tracking was lost, by relocalizing from the bag-of-words recognition database. The pose estimation is optimized by projecting the local map of keyframes containing covisible map points into the current frame. The keyframe with the most covisible map points is designated as the reference keyframe. If the current frame's image is sufficiently different from that of the reference keyframe, it is inserted into the covisibility graph as a new keyframe in the mapping thread. Map points that have been determined to be non-trackable or erroneously triangulated are culled. Otherwise, new map points are triangulated from matching ORB features in connected keyframes. Local bundle adjustment is run to optimize the current keyframe along with the connected keyframes in the covisibility graph and all map points belonging to those keyframes. Redundant keyframes are then culled.

The loop closing thread utilizes the bag-of-words approach to query the recognition database, find similar images, and insert new edges into the covisibility graph for loop closures. The loop closure error is then propagated throughout the graph by optimizing the essential graph and transforming all map points according to its keyframe's correction. A new version of the algorithm, ORB-SLAM2, was released in 2017 [5]. ORB-SLAM2 adds in a full bundle adjustment step over all keyframes and map points in a separate fourth thread. It also extends the algorithm to accommodate monocular, stereo, and RGB-D cameras. ORB-SLAM and ORB-SLAM2 are both open-source.

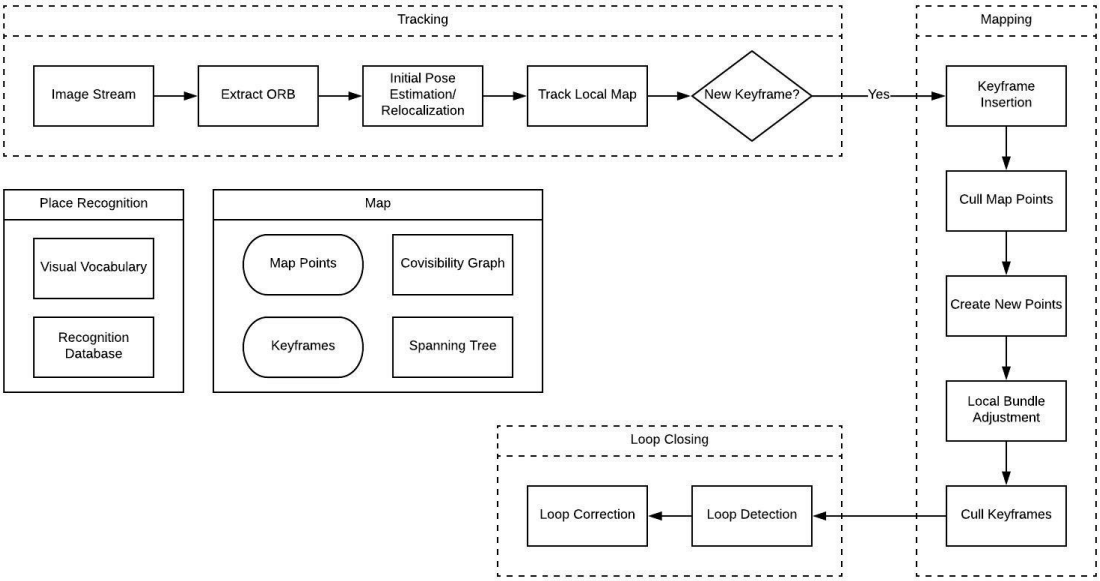

Figure 10. ORB-SLAM

SVO is a hybrid VO algorithm by Forster et al. that was first released in 2014 [3]. Figure 11 gives an overview of the algorithm. SVO consists of a motion estimation or tracking thread and a mapping thread. In the motion estimation thread, an initial estimate of the camera pose is found through sparse model-based image alignment in which a direct method is used to minimize the photometric error with reprojected feature patches from the previous image. The reprojected features in the new image are then aligned with respect to the rest of the map by optimizing the 2D pixel locations to minimize the photometric error with the reference feature patch in the keyframe. By adjusting each individual reprojected feature locations in the new frame, the feature alignment step violates epipolar constraints. In order to correct this, SVO performs a bundle adjustment through the pose and structure refinement step, optimizing the camera pose again but this time by minimizing the photometric error with respect to optimized feature patches. If the mapping thread receives a keyframe, it splits the image into fixed-size cells and extracts the FAST corners with

the highest Shi-Tomasi score in each cell. Depth filters are initialized for new features with high uncertainty. New regular frames are used to update these depth filters using a Bayesian method and once the variance is sufficiently low, the corresponding 3D point is inserted into the map to be used for motion estimation. By only extracting feature points during keyframes and using direct methods to calculate the camera pose for every frame, SVO is able to achieve high processing speeds. The authors also presented SVO 2.0 in 2017 [21]. This extends the original SVO algorithm to large FOV cameras, multi-camera systems, IMU incorporation, and the additional use of edges for feature alignment. Although the original SVO algorithm is open-source, SVO 2.0 is only available as a pre-compiled binary.

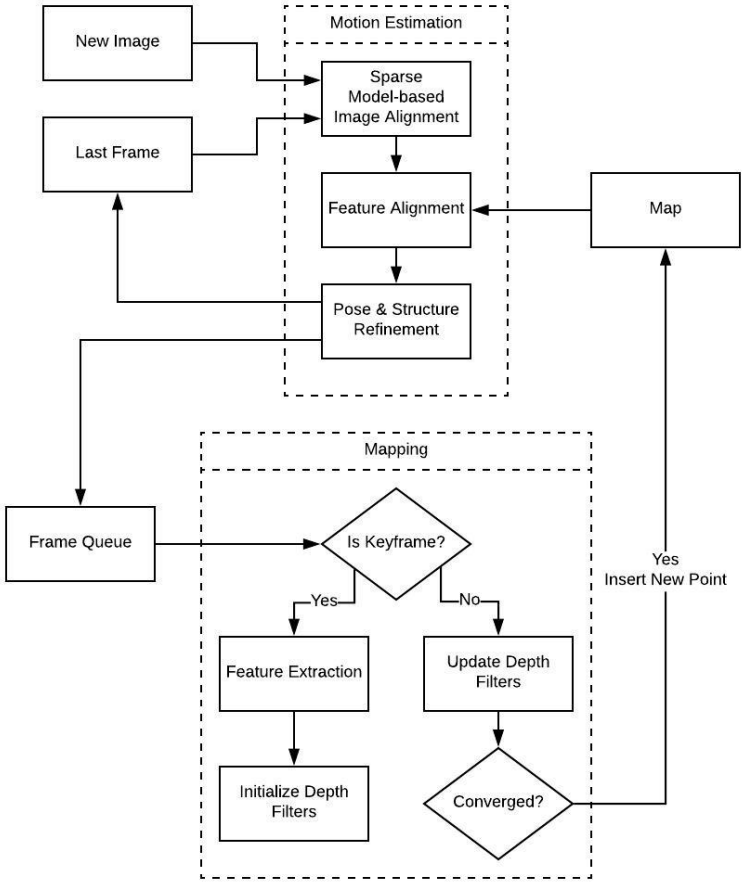

Figure 11. SVO

SVO has been used as a baseline algorithm in multiple follow-on works by the Robotics and Perception Group in the University of Zurich, Switzerland. REMODE is a monocular dense reconstruction algorithm by Pizzoli et al [50]. This algorithm uses a similar probabilistic Bayesian approach as SVO to build dense depth maps from a monocular camera. However, REMODE is highly parallelizable and designed to use a GPU for real-time dense reconstructions at 30 Hz. SVO is used as a component in REMODE to estimate the camera pose. Forster et al. used both SVO as well as a modified version of REMODE on a quadrotor to develop an onboard real-time elevation mapping algorithm [51]. In this system, SVO pose outputs from a downward facing camera are fused with IMU measurements using an Extended Kalman Filter (EKF) multi-sensor fusion (MSF) package to obtain scaled trajectory estimates. REMODE is scaled down by initializing depth filters for blocks of pixels instead of every single pixel, thereby allowing onboard computation on a CPU. The depth map built by REMODE is then used to build a 2D elevation map that remains localized around the quadrotor and is used for autonomous landing. Alternatively, Faessler et al. used both SVO and REMODE on a quadrotor with a downward facing camera to autonomously execute a trajectory while providing a dense 3D map of the traversed area in real-time [31]. SVO outputs are again fused on an onboard CPU with IMU measurements through an EKF MSF package and the results are used to provide inputs at 50 Hz to the low-level flight controller on the quadrotor. SVO outputs are sent over WiFi at 5 Hz to a ground-station running REMODE to generate live dense maps. This system was tested on a 140 m trajectory at an operating altitude of 20 m in an outdoor firefighter disaster mock-up site in Zurich.

In recent years, a completely new approach to the VO problem has been in development applying deep learning and artificial neural networks. Wang et al. developed DeepVO which utilizes deep recurrent and convolutional neural networks to calculate

pose estimates from a set of images [52]. This method entirely avoids the need for geometric optimizations or even camera calibrations as opposed to feature-based and direct methods. It also allows recovery of the absolute scale with no prior knowledge or additional input as this is learned while training the neural net. Li et al. proposed UnDeepVO which uses unsupervised deep learning to allow training its neural net on unlabeled datasets [53]. Unlabeled datasets are those that are not formatted with ground truth data. This is useful since obtaining ground truth is usually a difficult and expensive operation, especially if it must be gathered and synchronized from multiple sources running at different rates. In order to recover scale, UnDeepVO must first be trained using a stereo camera configuration before using it as a monocular VO system.

Several public VO and VSLAM datasets with synchronized truth data are available and are used by many works to compare and report algorithm performances. The KITTI dataset contains data from two stereo camera setups as well as a rotating 3D laser scanner captured from a station wagon driving through traffic [54]. The truth trajectory data is gathered from a high-precision GPS/IMU system. The laser scanner and image data are collected at a rate of 10 frames per second. The TUM RGB-D dataset consists of color and depth images collected from a Microsoft Kinect sensor at 30 frames per second in an indoor office environment from either a handheld configuration or on a wheeled robot [55]. Truth trajectory data is obtained from an external motion capture system. The TUM monoVO dataset contains 50 sequences of monocular camera data exploring both indoor and outdoor environments from a handheld configuration [46]. Camera image rates range from 20 to 50 frames per second. The dataset provides both geometric and photometric camera calibrations for all sequences. Instead of measuring the ground truth using external sensors, this dataset has all sequences start and end at the same position and uses LSD-SLAM

to generate the truth data after conducting a large loop closure. Therefore, the ground truth is not perfectly accurate. The European Robotics Challenge (EuRoC) Micro Aerial Vehicle (MAV) dataset provides stereo visual and inertial data from a hexrotor flying in an indoor industrial environment [56]. Camera data is provided at 20 Hz and ground truth data is recorded using a laser tracking system and a motion capture system. The laser tracking system provides millimeter accuracy at 20 Hz while the motion capture system provides pose measurements at 100 Hz. The Zurich Urban MAV dataset provides monocular image data collected from a quadrotor flying outdoors over the streets of Zurich, Switzerland at an altitude of 5 to 15 m over a 2 km trajectory [57]. Ground truth data is calculated by appearance-based topological localization and VSLAM algorithms.

The virtual environment created in this work aims to solve some critical limitations of these datasets. The first limitation is that the datasets are aimed at handheld platforms, ground-based vehicles or small quadrotors which operate under different environments from fixed-wing SUAS. A second limitation is that obtaining accurate ground truth data at high frame rates in the real world is a difficult task. However, perfect truth data is readily available in the virtual environment. Finally, the virtual environment allows configuration of SUAS parameters and trajectories to allow fast and easy test flights in a multitude of scenarios that are not limited to those in the datasets.

Ellingson et al. recently published a work on visual-inertial odometry for fixed-wing aircraft [33]. In this work, a modified version of a previously proposed relative navigation architecture is used. This architecture consists of a relative front end running a multi-state constraint Kalman filter (MSCKF) for state estimation and a global back end that maintains a pose graph and conducts optimizations as well as loop closures [58]. This algorithm is tested in a simulation environment built using

the Gazebo robot simulator. The simulated aircraft flies over a cityscape image at 11 m/s at an altitude of 50 m. However, the MSCKF filter implementation is not yet fast enough to run under real-time constraints and the frame rate of the virtual aircraft camera is limited to three frames per second.

Carson investigated the use of a Kalman Filter to implement visual-inertial odometry on a fixed-wing aircraft [32]. In this work, four VO algorithm variants were created as shown in Figure 12. These algorithms are used along with SVO in a Kalman filter-based solution fusing additional inputs from an IMU, barometer, and terrain elevation data. Frame-by-frame velocity calculations are output from all VO algorithms into the Kalman filter. The created VO variants are all frame-by-frame algorithms with no bundle adjustment or local mapping. They vary in their tracking method as well as their method of calculating rotation. Tracking is conducted either through feature detection and brute force matching between two images of AKAZE features [59] or by dividing the images into uniform grids and tracking pixel features using Lucas-Kanade tracking of optical flow [60]. The rotation matrix is calculated either through the essential matrix from the imagery or completely supplanted by the filtered INS measurement. The output velocity scale calculation relies on the assumption that the camera is facing straight down throughout the entire trajectory and requires continuous updates of the filtered altitude solution from the barometer, IMU, and terrain elevation data.

One of the limitations of Carson's work is that the created VO algorithms rely on the assumption that the camera is always perfectly face-down to calculate the velocity scale on every frame, limiting their flexibility during pitches or rolls. The comparison against SVO also fails to account for SVO's bundle adjustment behavior since SVO is incorporated into the Kalman filter by calculating the output velocity from the difference in position of two frames while never updating the local map points. Also

in this work, SVO is tested on imagery collected at a rate of five frames per second. This is a suboptimal condition since the algorithm relies on higher frame rates for faster convergence and greater robustness [21].

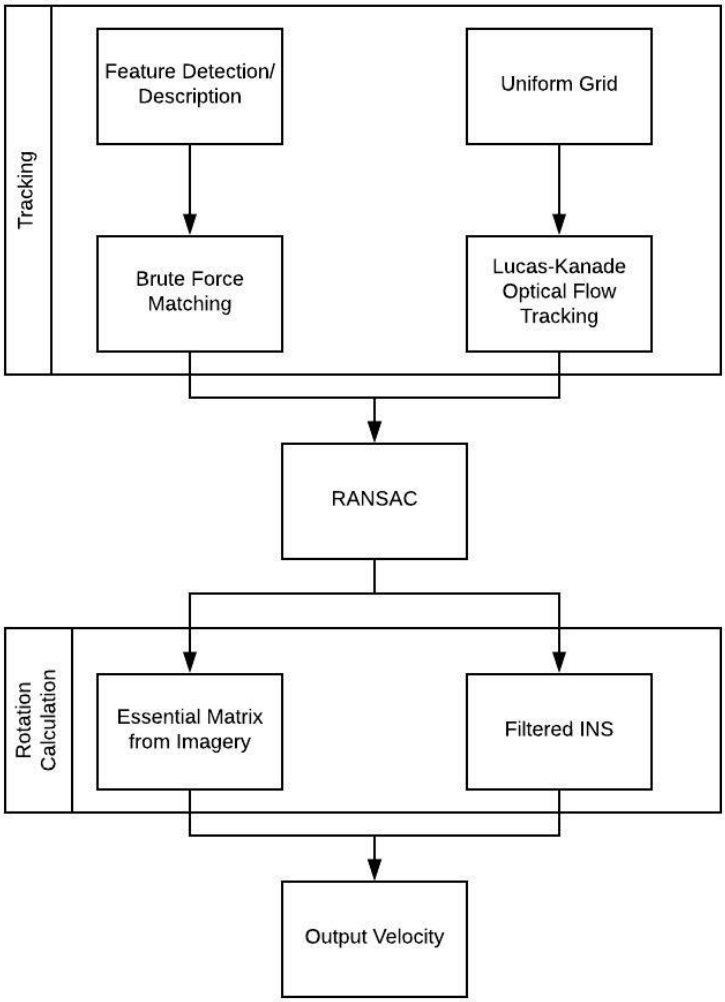

Figure 12. Carson VO Variants used in [32].

2.7 Summary

This chapter introduced core concepts regarding autonomous SUAS navigation and visual methods. It presented the formulations for the geometric camera cali-

bration matrix, epipolar constraint for feature-based VO and the photometric error optimization for direct methods. Finally, it presented works in both VO and VSLAM that have made contributions to the field.

III. Methodology

This work makes the following contributions in VO: a virtual environment for testing visual navigation algorithms and an analysis of current state-of-the-art VO algorithms applied to fixed-wing SUAS platforms. This chapter provides a detailed description of all modules built for both virtual and real-world testing. This chapter also presents the experimental design and methodology used to analyze and compare the performance of the VO algorithms on a fixed-wing SUAS.

3.1 Dependencies

This section describes the core software dependencies used to build the different modules. All modules are built and executed on an Intel i7, 2.8 GHz quad-core laptop running an Ubuntu 16.04 operating system (OS). All modules are programmed in C++ and requires compiler support for C++14 or greater.

3.1.1 Middleware.

A core issue for robotic systems is inter-process communication (IPC) since most systems require the development and use of multiple subsystems running as separate processes or nodes. Several open-source middlewares have been developed to address this problem and one of the most popular is the Robot Operating System (ROS) [61]. IPC in ROS is accomplished through a message-passing system where language-agnostic message types can be built and used to generate the appropriate language-specific data structures and files to be imported by nodes for communication. Messages are passed from publisher to subscriber nodes that are on the same topic. Key design features in ROS are that all communications between nodes are managed by a centralized node and that the principal protocol used is the Transmis-

sion Control Protocol (TCP). This makes ROS less than ideal for building distributed systems with potentially unreliable connections between nodes. However, ROS features an extensive mature library beyond IPC with a large open-source community, provides a centralized parameter server and launch system for easy management and distribution of data among nodes, and maintains a build system augmenting CMake for support in importing ROS packages and libraries.

Another open-source middleware is the Lightweight Communications and Marshalling (LCM) library [62]. This was designed by students at MIT to offer the bare essentials of just IPC. LCM operates using a similar system as ROS, passing messages from publisher nodes to subscriber nodes that are on the same channel. Communication between nodes is accomplished through the User Datagram Protocol (UDP) in a decentralized network, making LCM more suitable for distributed systems. However, LCM lacks the presence of ROS's large open-source community and libraries. LCM also lacks a centralized parameter server, requiring alternative methods to distribute parameters to disparate nodes.

Both ROS and LCM are used in multiple modules built for this work. ROS is used for its launch system and parameter server to be able to correctly initialize and launch multiple nodes. It is also used to take advantage of the MAVROS library, an extensive and mature library for communicating with the ArduPilot autopilot and simulator. LCM is used for compatible communication with systems being developed at the AFIT Autonomy and Navigation Technology (ANT) Center. ROS Lunar and LCM version 1.3.1 are used in this work.

3.1.2 AftrBurner Engine.

All virtual environment components are built using the AftrBurner engine, a cross-platform visualization engine written in C++ and the successor to the STEAMiE

educational game engine [63]. The visualization engine contains submodules for creating and reading virtual camera data as well as generating scaled real-world terrain models using United States Geological Survey (USGS) elevation data and satellite imagery. The AftrBurner engine also contains formulations for modeling the World Geodetic System (WGS) 84 model to accurately visualize GPS coordinates. Figure 13 shows an example terrain model of the Grand Canyon created through the AftrBurner engine mapped against GPS coordinates and USGS elevation data. Limiting factors include the available resolution of terrain and satellite imagery and the image distortion incurred from mapping a 2D overhead image onto a 3D model. This distortion is more pronounced in extreme elevation changes as shown on the cliffs and walls of the virtual Grand Canyon.

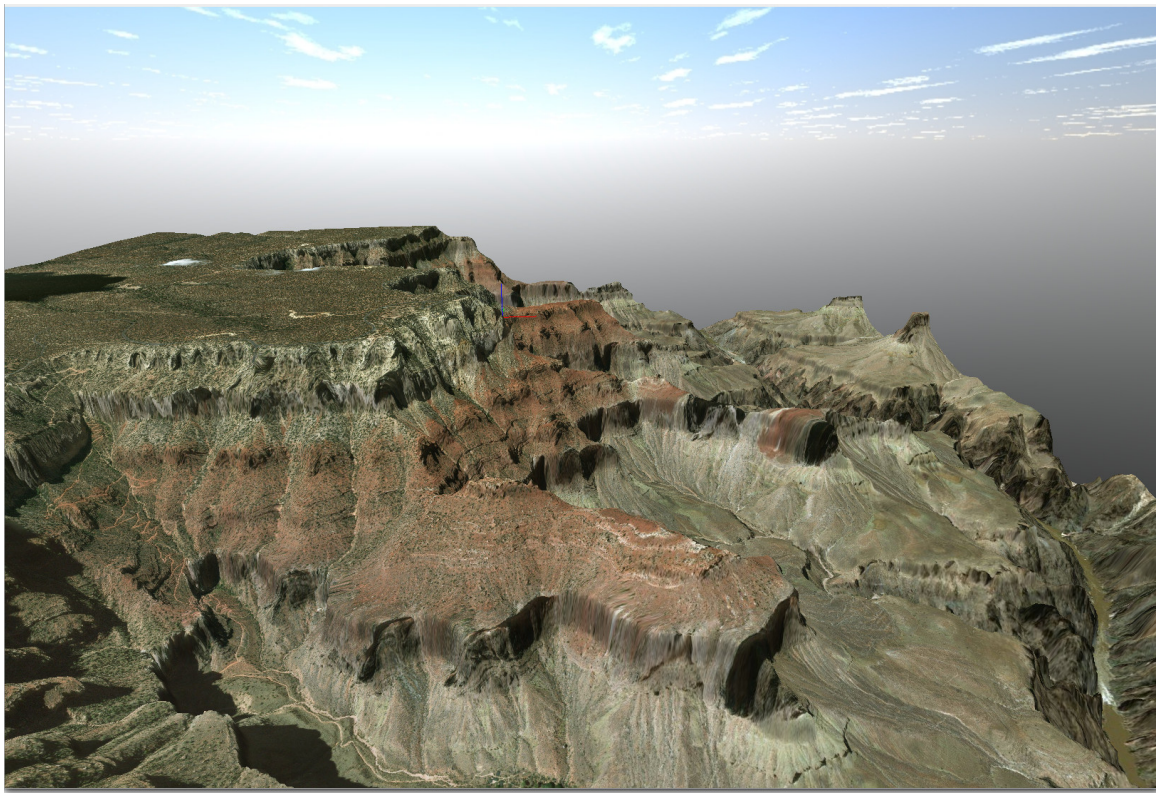

Figure 13. Virtual Grand Canyon

3.1.3 Qt.

Qt is a cross-platform Software Development Kit (SDK) for building Graphical User Interface (GUI) applications. This is used to augment some of the modules built with the AfterBurner Engine to provide intuitive controls. Qt 5.10.1 is used for all applications.

3.1.4 OpenCV.

The Open Source Computer Vision library (OpenCV) is a C++ library that is used for image processing applications including all of the VO algorithms used in this work [64]. OpenCV is also used for its camera calibration functions. OpenCV version 3.4.1 is used for all applications.

3.1.5 ArduPilot.

The ArduPilot autopilot software is used to control the aircraft in both the virtual environment and the real world. ArduPilot uses an EKF to fuse sensor input and provide low-level control for various autonomous vehicles including rovers, submarines, copters, and planes. Raw and fused sensor data can be read from the ArduPilot. The ArduPilot uses the MAVLink protocol to transmit and receive data. The MAVROS library is used to connect with the ArduPilot through a ROS node and read MAVLink data. ArduPilot provides a Software In The Loop (SITL) program which allows ArduPilot software to be run on a computer and simulate virtual autopilots for vehicles. This is used to generate simulated trajectories for a fixed-wing aircraft exhibiting realistic flight dynamics. A ground control station (GCS) program is required to give commands and communicate with the ArduPilot. The GCS used for controlling the simulated aircraft is QGroundControl since it is available on Linux, allowing simultaneous execution of all modules on the same laptop computer. The

GCS used for controlling the real-world aircraft is Mission Planner since that is the standard GCS used in the ANT Center.

3.1.6 Vimba SmartCables Driver.

The Vimba SmartCables Driver is a LCM wrapper built around the Vimba Software Development Kit (SDK) for interfacing with Allied Vision cameras. The driver is maintained by the ANT Center and is tested with Vimba 2.1.3.

3.1.7 Dependency Conflict.

The AftrBurner engine requires the Boost library with version 1.66 or greater. However, the precompiled ROS library that is installed by the Linux package manager, Advanced Package Tool (APT), in Ubuntu 16.04 is compiled against an older incompatible version of Boost. This causes a problem when using the AftrBurner engine with ROS that prevents transmitting and receiving image messages through the ROS image_transport package. This can be corrected by uninstalling all Boost libraries older than version 1.66 and compiling the entire ROS library from source against the installed Boost library version 1.66 or higher. Some ROS modules may exhibit compilation errors upon which those modules can be removed if deemed non-essential. An example of this is OpenCV, as that should be compiled separately from the ROS library. For other ROS modules not included in the core package such as MAVROS, the source code for the package and each of its ROS dependencies must be manually downloaded from their respective git repositories and compiled. Older versions of Boost that may be installed as dependencies of other APT packages after compiling the AftrBurner engine and ROS may cause applications to link against the wrong version of Boost and cause the same errors during image transmission. Removing the older versions of Boost will correct this error.

3.2 System Design

3.2.1 VO Nodes.

The three current state-of-the-art algorithms tested are DSO [4], SVO [3], and ORB-SLAM2 [5]. SVO is used instead of SVO2 since the source code for SVO2 is not currently open-source. ORB-SLAM2 is allowed for comparison against the other VO algorithms by disabling the loop closure thread so that it essentially functions as a feature-based VO algorithm.

Although all of the algorithms provide example ROS wrappers and instructions for implementing ROS nodes, SVO provides the most portable library and full-featured functioning ROS node along with supporting parameter and launch files. DSO and ORB-SLAM2 require some restructuring and rewriting of their CMake files to make their dependencies more portable and the projects more accommodating for encapsulation by a wrapper node without introducing circular dependencies. Therefore, the DSO and ORB-SLAM2 modules are modified to conform to a similar project structure as SVO.

Figure 14 shows the standardized ROS and LCM node for each of the algorithms along with the parameters used by the nodes which are provided through the ROS parameter server. The solid arrows show the input and output ROS topics that the node subscribes and publishes to. The dotted arrow shows the input LCM channel that the node subscribes to. The node primarily interacts with other modules through ROS messages, taking in raw camera images as input and outputting the camera pose data relative to the initial frame and an image showing the processed state of the input. However, the node also accommodates LCM input images for flexibility in interfacing with ANT Center systems. Currently, the only supported camera model that is common to all three algorithms is the pinhole model. The cam_width and cam_height parameters hold the image pixel width and height. The cam_fx and cam_fy parame-

ters correspond to the camera focal lengths while the cam_cx and cam_cy parameters correspond to the image center in the camera matrix. The rest of the parameters correspond to the camera's distortion coefficients. Individual VO algorithms also contain separate parameters for detailed settings specific to the algorithms.

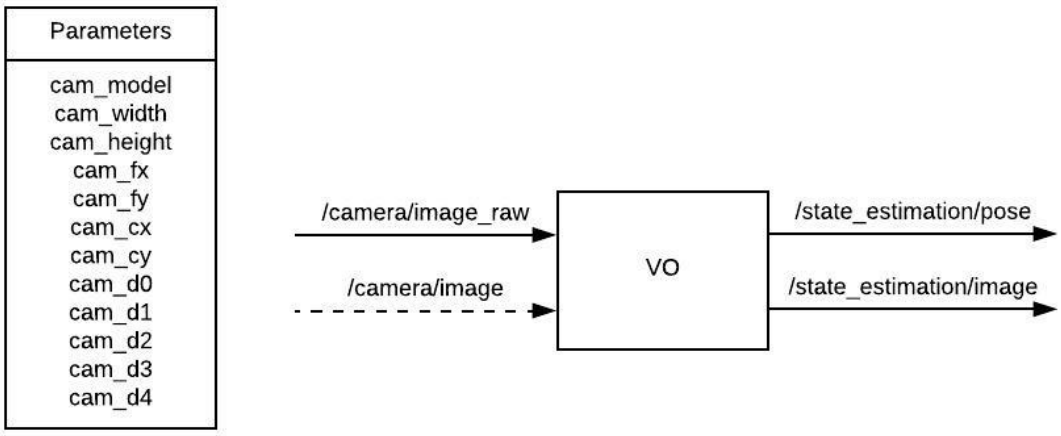

Figure 14. VO ROS/LCM Node

3.2.2 Camera Calibration.

A GUI-based calibration module is used to calibrate both real-world and virtual cameras. Figure 15 shows the camera calibration modules. The Vimba SmartCables Driver is used to interface with Allied Vision Cameras and collect images. A virtual monocular calibration module provides imagery of a virtual checkerboard from a single camera with adjustable settings for FOV, aspect ratio, and image resolution as shown in Figure 16. Image data is communicated through LCM to allow interchangeability of image sources for the calibration GUI. The calibration GUI currently only provides calibrations for the pinhole model. The GUI accepts checkerboard properties as input and displays the live stream of raw image data allowing the user to capture desired images for calibration. The GUI then detects the checkerboard corners in the image,

displays the results to the user, and stores its measurements until the user wishes to compute the calibration matrix and distortion coefficients as shown in Figure 17. Once the calibration is complete, the GUI displays the raw image stream side-by-side with the undistorted result.

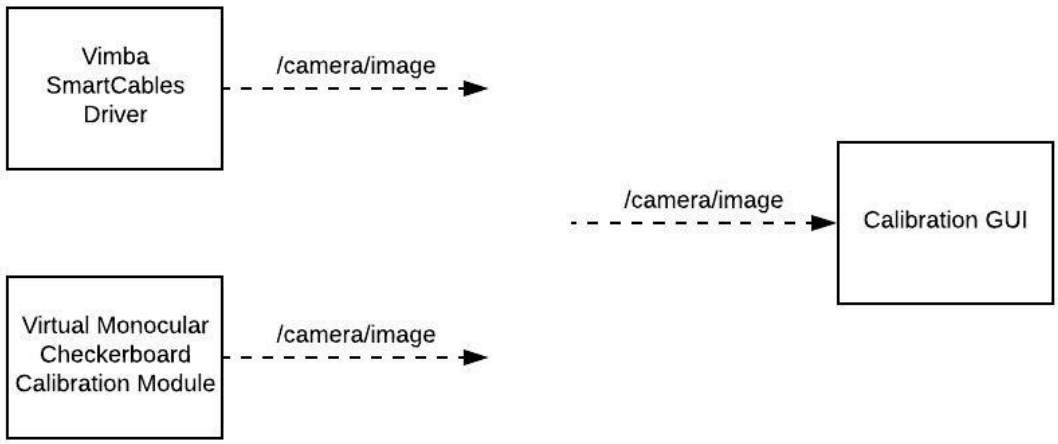

Figure 15. Camera Calibration Modules

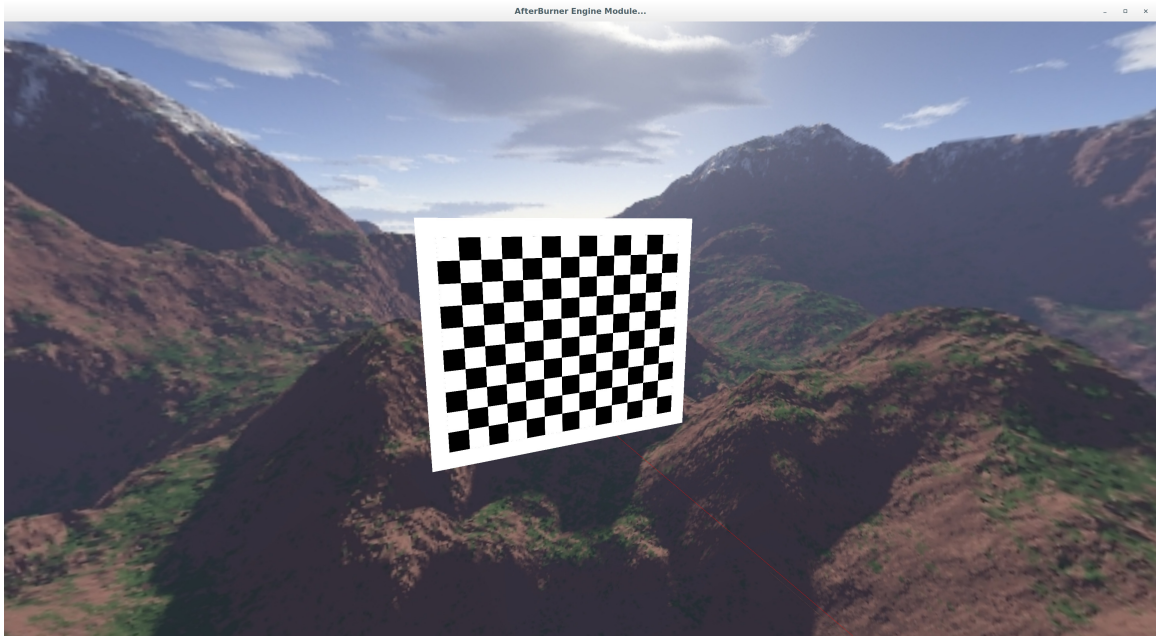

Figure 16. Virtual Monocular Calibration Module

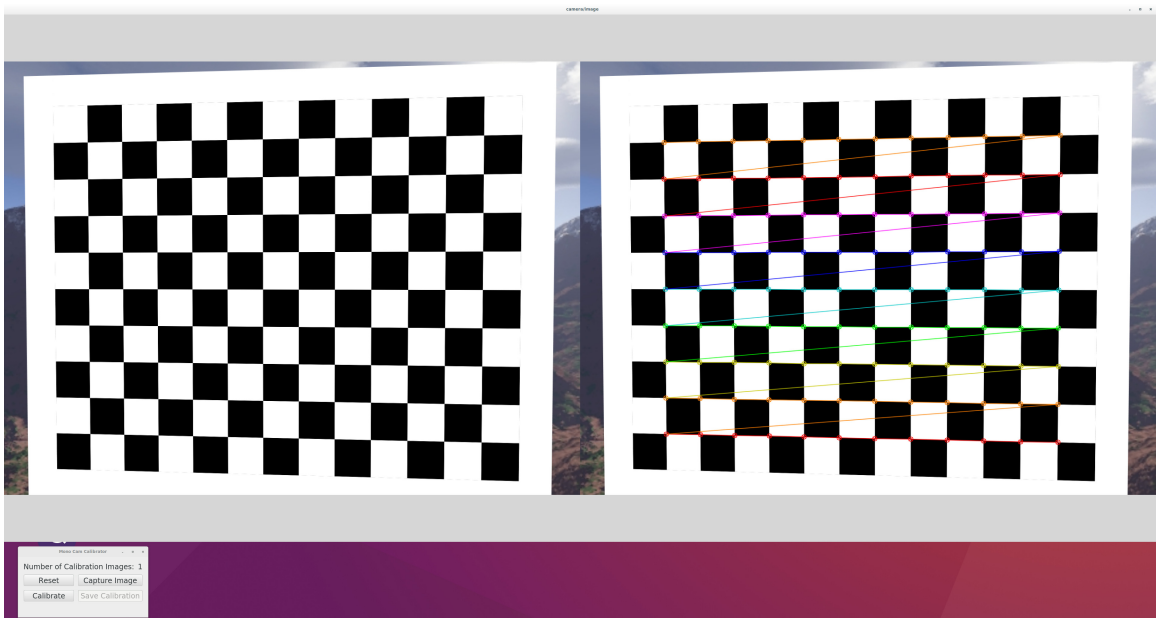

Figure 17. Camera Calibration GUI

3.2.3 Autonomy Simulator.

The autonomy simulator is a flexible and modular virtual environment that allows both simulated and real-world flight profiles to run in real-time over GPS terrain mapped satellite imagery. It generates perfect truth data and transmits virtual camera imagery at 31 fps. It also reprojects estimated pose trajectories from a state estimation algorithm in real-time. The autonomy simulator is able to record and play back trajectories with microsecond accuracy. Truth trajectories are recorded either from simulated or real-world flights and multiple iterations of VO algorithms can be run and recorded on the same truth trajectory. Figure 18 shows an overview of the autonomy simulator and its interaction with other modules.

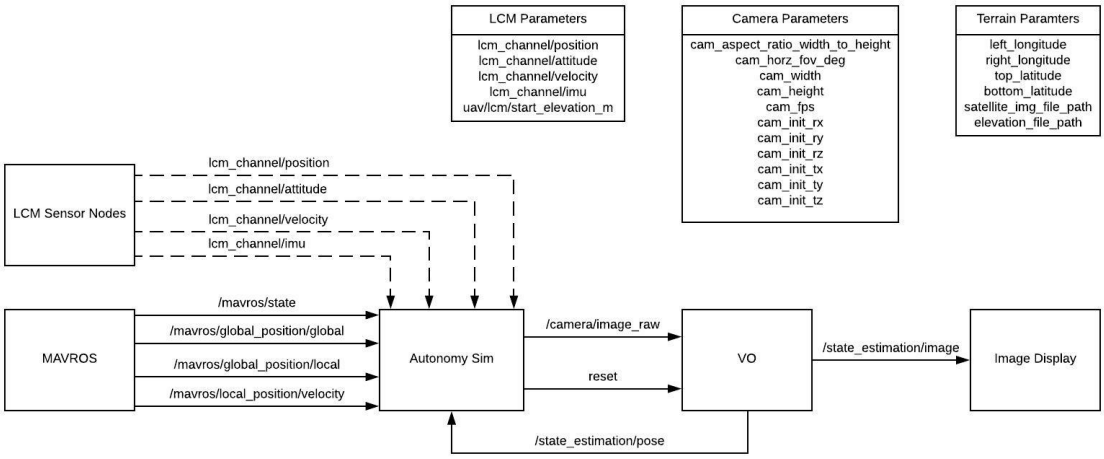

Figure 18. Autonomy Simulator Overview

The autonomy simulator interprets ROS inputs from MAVROS to interact with SITL or an actual Ardupilot autopilot to display the current state of the SUAS. The /mavros/state topic notifies when the autopilot changes modes such as when it is connected, armed or takes off. The /mavros/global_position/global topic provides the GPS location with latitude and longitude in degrees and an altitude above the WGS-84 ellipsoid in meters. /mavros/global_position/local provides the local orientation in a frame with coordinate axes facing East, North, and up away from the center of the Earth (ENU). /mavros/local_position/velocity provides the linear velocity in meters per second (m/s) in the ENU frame and the angular velocity in radians per second in a frame with coordinate axes facing front, left, and upwards relative to the aircraft's body.

The autonomy simulator also accepts LCM inputs to display the current state of the SUAS. The LCM inputs are primarily used to interface with the SUAS developed in the ANT Center. The lcm_channel/position channel provides the GPS position with latitude and longitude in radians and an altitude measurement relative to the starting altitude when the SUAS is first initialized in meters. lcm_channel/attitude

provides the orientation in a frame with coordinates facing North, East, and downwards towards the center of the Earth (NED). lcm_channel/velocity provides the linear velocity in m/s in the NED frame. lcm_channel/imu provides the angular velocity in deg/s in a frame with coordinate axes facing front, right, and downwards relative to the aircraft's body.

The simulator outputs imagery from the virtual SUAS camera to a VO node through a ROS topic. The simulator also uses a ROS service call to be able to reset VO algorithms. The VO node outputs its pose estimate relative to its starting pose back into the simulator to be displayed alongside the current pose of the actual SUAS. The processed VO images can be displayed by a separate node. The simulator requires parameters to set the individual LCM input channels. Parameters must also be provided to set the virtual SUAS camera properties to include the aspect ratio, FOV, resolution, frame rate, and the camera orientation and location relative to the center of the SUAS model. The frame rate of the virtual camera can be adjusted with a maximum rate of 31 frames per second. Finally, terrain parameters must be provided to set the boundary latitudes and longitudes as well as the satellite imagery and elevation files to load from.

The MAVROS inputs are primarily used to interface with SITL and generate simulated flight trajectories. QGroundControl is used to plan detailed trajectories and set SUAS parameters such as airspeed and maximum roll rate. Figure 19 shows a simulated flight over Camp Atterbury, Indiana with QGroundControl in the upper left window showing the planned trajectory with live updates from SITL, the autonomy simulator in the right window showing the live 3D trajectory with a red trailing ribbon drawn to scale and the live raw image feed in the left window from the virtual camera which faces downward relative to the SUAS.

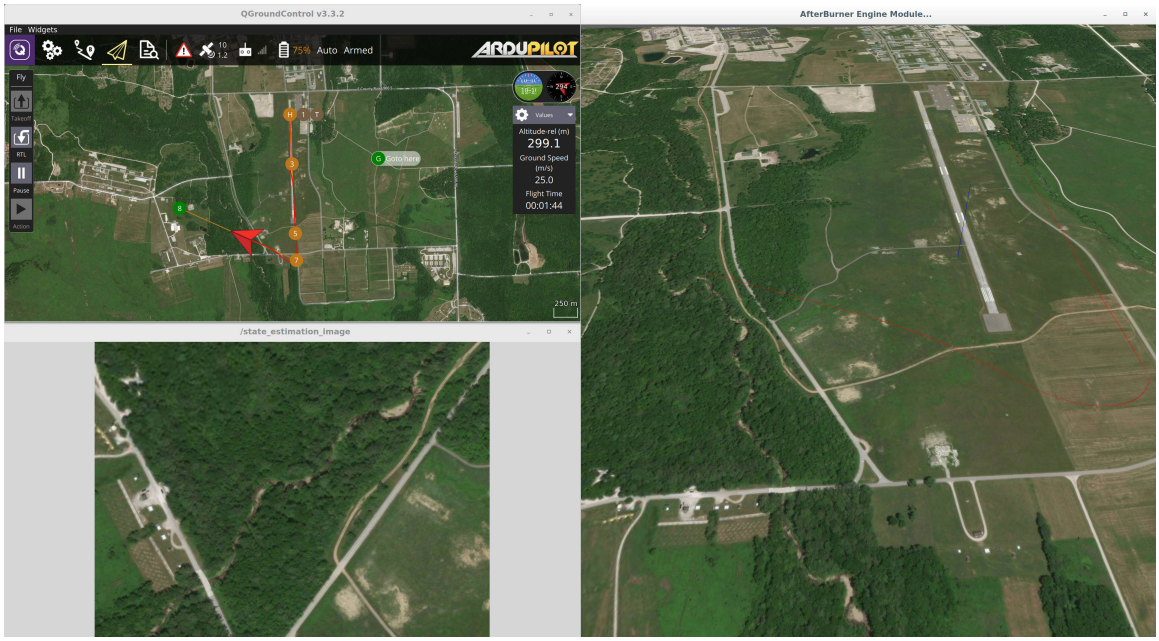

Figure 19. Simulated Flight Trajectory. QGroundControl in the upper left window, autonomy simulator in the right window and the live virtual camera feed in the lower left window.

Figures 20, 21 and 22 show examples of running DSO, ORB-SLAM2, and SVO respectively on simulated flight trajectories and reprojecting the pose estimates back into the autonomy simulator. The left windows show the processed images from the VO algorithms with the tracked pixels or features in the current image. The right windows show the simulator with the truth trajectory represented by the SUAS with the red trailing ribbon and the estimated trajectory by the SUAS with the blue trailing ribbon.

Since all of the monocular VO algorithms estimate the trajectory up to a relative scale, the absolute scale for all VO runs in this work are determined by the initial altitude of the SUAS when the VO algorithm is started or reset. This relies on the assumption that the camera is facing perfectly downward when the algorithms are initialized so that the depth of the first feature points or pixels are the same as this altitude. Therefore, an initial leg is included in every flight plan where the

SUAS flies on a straight and level path at 20 m/s to conduct VO initializations. This method of determining absolute scale is susceptible to unconstrained growth in error as the trajectory progresses but is sufficient for this work in order to strictly compare the core VO performances of each algorithm. Future work should incorporate measurements from additional sensors such as an IMU to determine the absolute scale continuously. However, since the IMU also experiences drift, a loop closure method must be employed along with the VO algorithm to reduce the drift in scale.

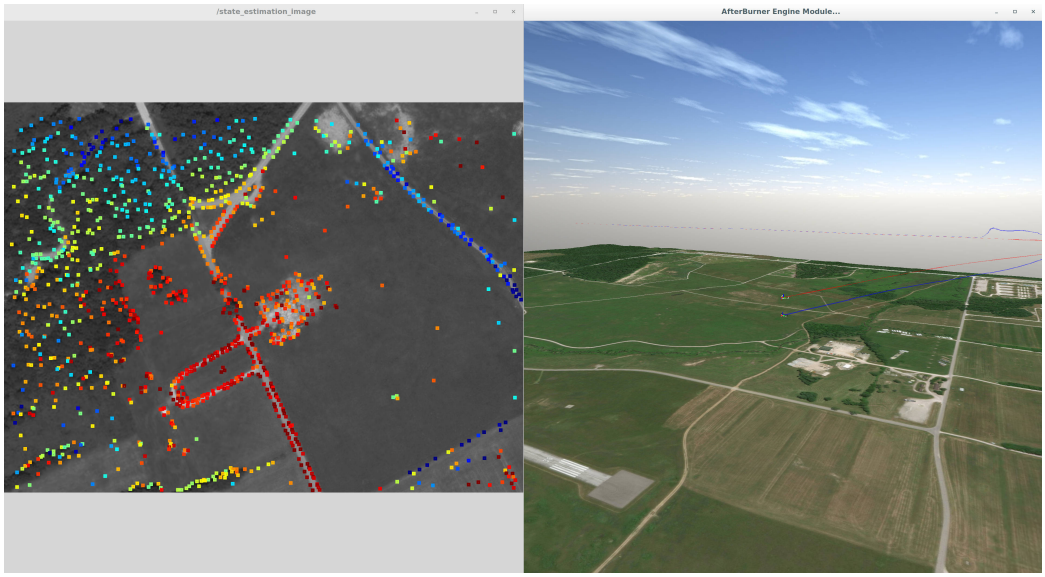

Figure 20. Simulated Flight With DSO. The left window shows the depth map of the latest DSO keyframe coloring pixels with the closest to the farthest depth from red to blue. The right window shows the autonomy simulator with the truth trajectory in red and the estimated trajectory in blue.

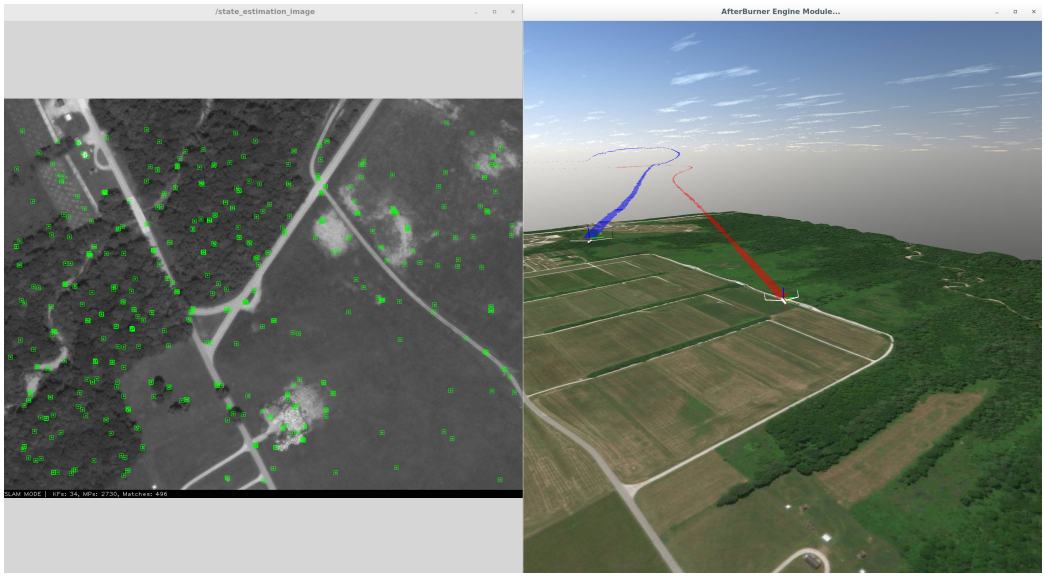

Figure 21. Simulated Flight With ORB-SLAM2. The left window shows the processed image with selected ORB points used by ORB-SLAM2. The right window shows the autonomy simulator with the truth trajectory in red and the estimated trajectory in blue.

Figure 22. Simulated Flight With SVO. The left window shows the processed image with selected FAST corners used by SVO. The right window shows the autonomy simulator with the truth trajectory in red and the estimated trajectory in blue.

3.2.4 Trajectory Viewer.

A separate trajectory viewer module is used to view multiple trajectories simultaneously. This module is primarily for playback of a single truth trajectory and multiple estimated trajectories from VO algorithms applied to that truth trajectory. Trajectory playback is synchronized across all trajectories, can be skipped to any point in time and sped up or slowed down. 3D plots of the trajectory and 2D plots for x, y, z, roll, pitch, and yaw data can also be generated. This allows detailed analysis and visualization of algorithm performances in all six degrees of freedom and to scale. Figure 23 shows an example of the trajectory viewer displaying multiple trajectories.

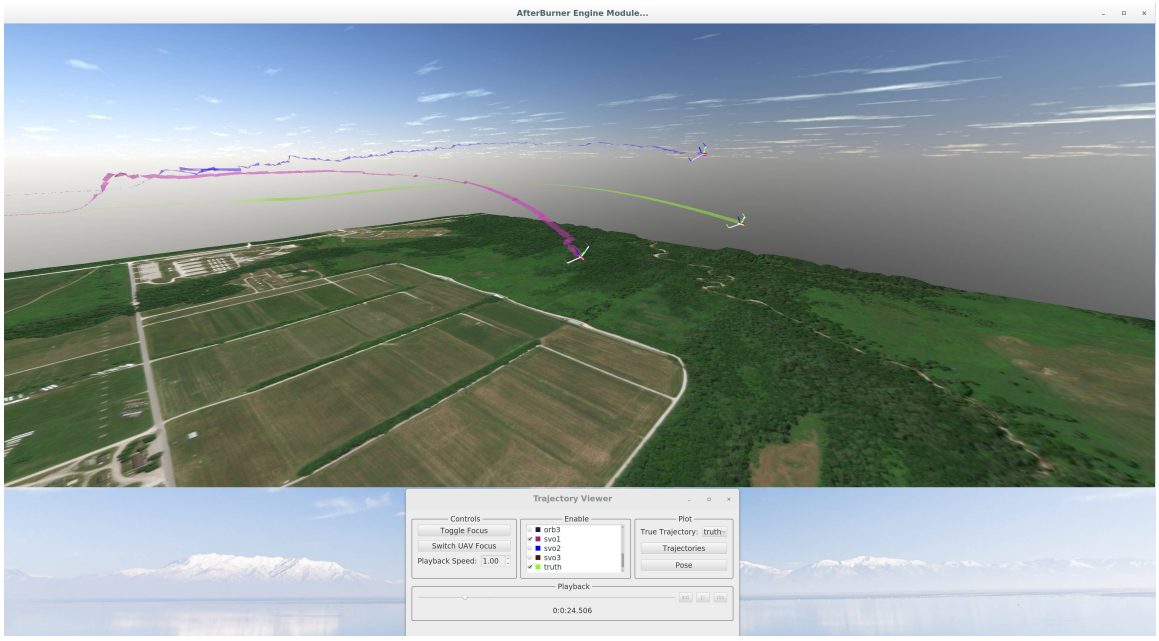

Figure 23. Trajectory Viewer

A current limitation of the trajectory viewer is that it can only load one dataset during each execution. Therefore, an additional trajectory plot viewer module is provided to allow analysis and plotting of multiple datasets in one process. It also provides more extensive plotting capabilities to be able to inspect individual plots in

more detail as well as calculate and print out the trajectory position and rotation errors. Figure 24 shows the controls for the trajectory plot viewer. The left-most window pane allows the selection of directories containing truth and estimated trajectories for a given flight. Selecting a directory auto-populates the next window pane with the trajectories to allow selective plotting. The truth trajectory file must also be identified since it is used to synchronize the estimated trajectory files to ensure that pose plots are generated against the same flight times. The next window panes contain controls for plotting and calculating the position and rotation errors for each of the trajectory files.

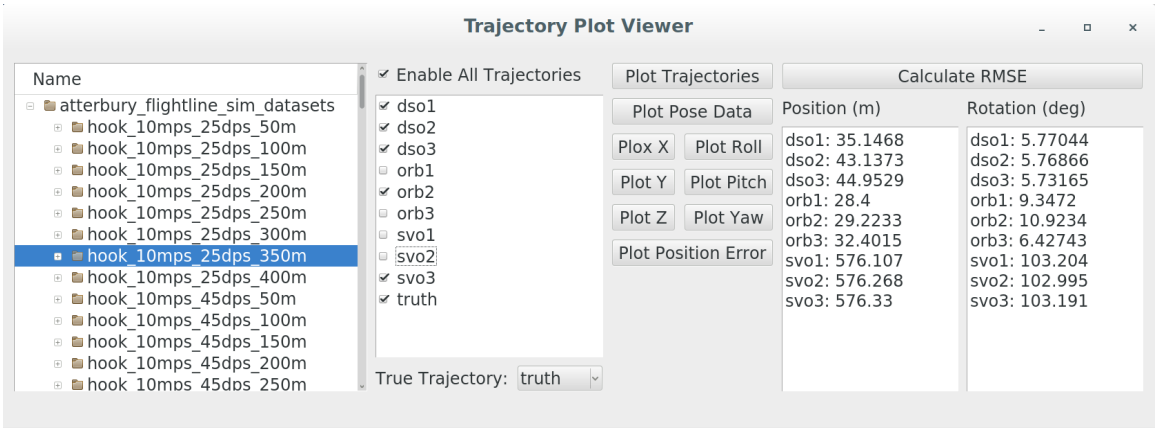

Figure 24. Trajectory Plot Viewer. From the left window pane to the right: directory tree for selecting datasets, trajectory file selection, plotting controls, error calculation.

3.2.5 Real-World Aircraft.

The VO algorithms are tested on data gathered from real-world flight tests using a 1/3-scale Carbon CUB aircraft. The CUB weighs 50 lbs and has a wingspan of 14 ft. A Pixhawk 2 autopilot is used for low-level control of the aircraft. All payload data is stored in LCM log files for playback and post-flight analysis. LCM log files were recorded in-flight on an onboard Intel NUC7 computer with an i7 processor.

Grayscale imagery was collected from a Prosilica GT1290 camera with a resolution

of 1280 × 960 pixels at 33 frames per second. The lens used with the camera has a sensor format of 1/2 in and is rated for an 84 deg FOV but the camera has a sensor format of 1/3 in, causing the actual image to have a slightly smaller FOV. The camera faces downward relative to the aircraft. Truth GPS and IMU data were provided by a Piksi navigation board at 10 Hz and 100 Hz respectively. All data are timestamped using a TM2000A Precision Time Protocol (PTP) server.

3.3 Experimental Design

This work tests and analyzes three of the current state-of-the-art VO algorithms on both simulated and real-world flight test data to determine the robustness and accuracy of the VO algorithms at different altitudes, speeds, and roll rates. The experimental results provide an understanding of which VO algorithms are viable for real-time fixed-wing SUAS operations and future incorporation into a larger SLAM scheme with multi-sensor fusion and loop closure for a complete GPS-denied navigation solution.

3.3.1 Assumptions/Limitations.

The following assumptions are used in this experiment:

1. The camera faces downward in the aircraft's body frame.
2. The virtual terrain image resolution is representative of real-world terrain when viewed from mission altitudes.
3. There is negligible error from the lack of parallax in the virtual terrain from non-elevation features such as trees when viewed from mission altitudes.
4. There is negligible distortion in the virtual terrain image from mapping the 2D image onto the 3D elevation map for Camp Atterbury.

5. VO algorithms are initialized when the SUAS is flying level and the camera is facing down. The altitude at initialization is equal to the absolute scale of the estimated trajectory.

Additionally, a limitation of this experiment is that a photometric camera calibration is not provided for DSO.

3.3.2 Simulation.

The tests in the virtual environment are conducted on the 3.3 km trajectory shown in Figure 25 over Camp Atterbury. The aircraft starts at the north end of the runway, flies south to reach mission altitude by waypoint 3, continues to a straight level flight at 20 m/s until waypoint 5 for VO initialization and then changes to mission airspeed to fly an oscillating pattern back north. Truth trajectories are recorded starting from between waypoints 3 and 5 once the aircraft has achieved steady level flight and ending once the aircraft has reached waypoint 13.

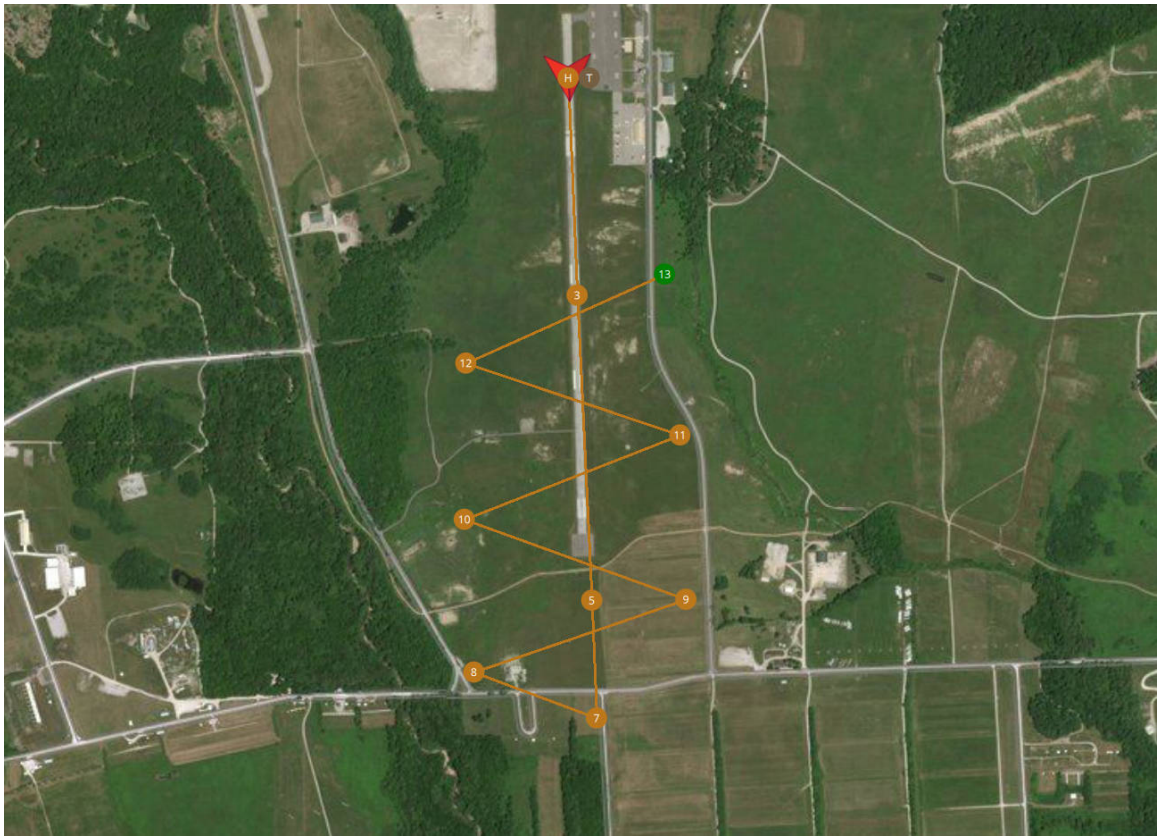

Figure 25. Simulation Test Trajectory

Experiment parameters are as follows:

- Airspeed (m/s): 10, 25
- Maximum roll rate (deg/s): 25, 45, 65
- Altitude: 50-400 m in 50 m intervals
- Image resolution: 1280 x 960
- Image aspect ratio (w/h): 1.333
- FOV (deg): 84.872
- Framerate (fps): 31, 5

3.3.3 Real-World.

Data from four real-world flight tests with the CUB flying between 20-30 m/s over Camp Atterbury with the following flight profiles are used to test the VO algorithms:

1. Box flight pattern as shown in Figure 26 at 450 m altitude
2. Box and then cloverleaf flight pattern as shown in Figure 27 at 400 m altitude
3. Grid flight pattern as shown in Figure 28 at 100 m altitude
4. Grid flight pattern at 250 m altitude

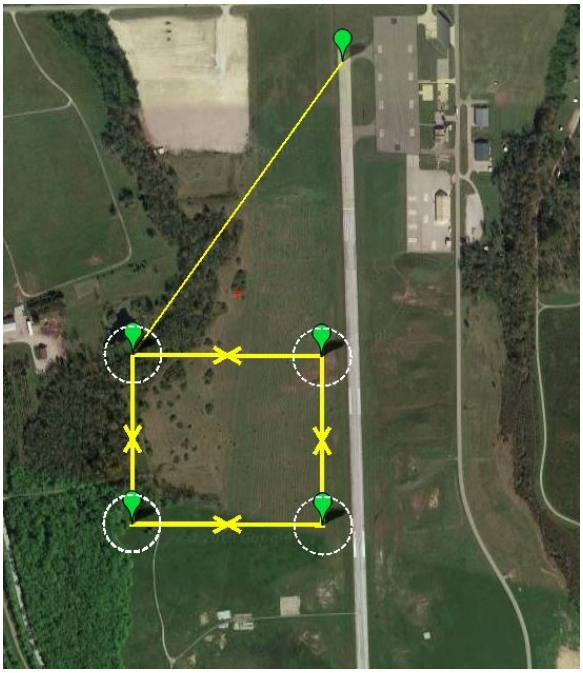

Figure 26. Box Flight Pattern

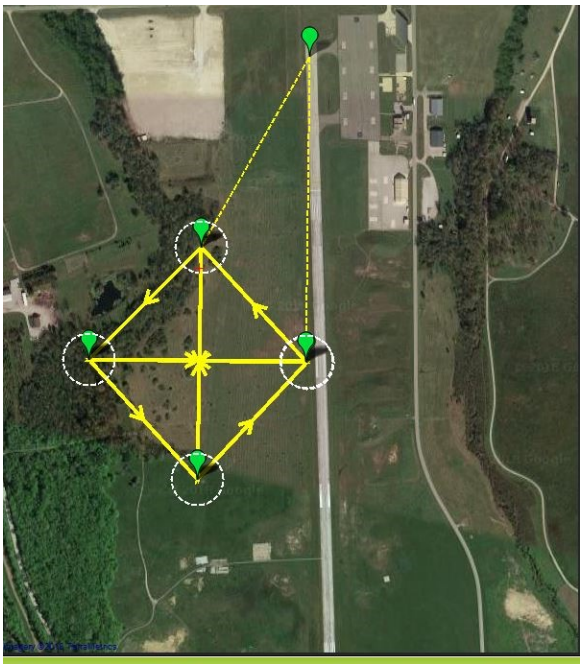

Figure 27. Cloverleaf Flight Pattern

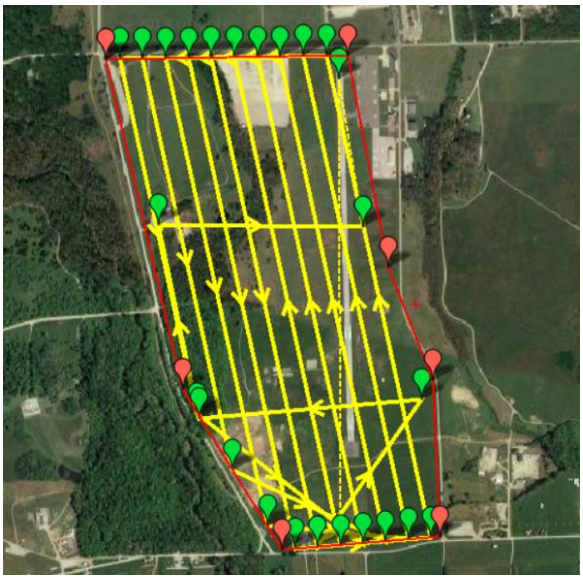

Figure 28. Grid Flight Pattern

3.3.4 VO Algorithms.

Each of the VO algorithms are tested using their default parameter settings. Additional tests are conducted by adjusting individual parameters specific to each algorithm. In DSO, setting_minGradHistAdd controls the image gradient threshold for candidate point selection in an image. Tests are run using the default value of seven and a lowered value of five to allow the selection of points with a lower gradient since many of the scenes viewed over Camp Atterbury's flightline have little texture [4]. The FAST score thresholds of ORB-SLAM2 and SVO are also tested at lower values for the same reason. ORBextractor.minThFAST in ORB-SLAM2 and triangMinCornerScore in SVO are tested at their default values of seven and a lowered value of two [21] [49].

SVO controls the creation of new keyframes through the kfSelectMinDist parameter which is the percentage of the average scene depth. A new frame is designated as a keyframe if it is farther than this distance from the previous keyframes. This is tested at its default value of 12% and a lowered value of 5% to create keyframes more frequently and increase the robustness of the algorithm under fast movements.

3.3.5 Processing Platform.

All VO algorithms are run on a laptop computer with a quad-core Intel Core i7-7500U CPU and a Nvidia Quadro M2200 GPU. The OS used is Ubuntu 16.04. Both the autonomy simulator and the VO algorithms are run simultaneously in real-time on the same laptop computer. For the real-world flight tests, flight test data including imagery are recorded onto LCM logs which are then used for real-time playback on the laptop computer. VO algorithms are executed on this recording and the truth data and VO results are projected onto the autonomy simulator in real-time on the same laptop computer.

3.3.6 Performance Metrics.

A VO algorithm's accuracy for a trajectory is measured by reporting the following metrics:

1. Root mean square error (RMSE) of the position error over x, y, and z in meters
2. RMSE of the rotation error over roll, pitch, and yaw in degrees

RMSEs for position and rotation will be calculated using the following equations:

$$RMSE_{position} = \sqrt{\frac{\sum_{i=1}^{n}(x_i' - x_i)^2 + (y_i' - y_i)^2 + (z_i' - z_i)^2}{n}} \tag{13}$$

$$RMSE_{rotation} = \sqrt{\frac{\sum_{i=1}^{n}(\phi_i' - \phi_i)^2 + (\theta_i' - \theta_i)^2 + (\psi_i' - \psi_i)^2}{n}} \tag{14}$$

where ϕ, θ, and ψ are the roll, pitch, and yaw, respectively. For a single data point at i out of n total data points, the variable $\mathbf{x}_i$ is the truth and $\mathbf{x}_i'$ is the output from the VO algorithm.

The VO's robustness for a trajectory is reported by the percentage of the total trajectory time that the VO algorithm is able to successfully track the trajectory without losing localization.

3.3.7 Summary.

This chapter described the system design for all virtual and real-world modules used to test the VO algorithms. All system dependencies were also detailed. The experiment was presented including the assumptions, flight patterns, and the parameters being tested. Finally, the performance metrics used to evaluate the VO algorithms were explained.

IV. Results & Analysis

This chapter presents the results obtained from the simulated and real-world experiments described in Chapter 3. The performance of each algorithm is analyzed according to its accuracy and robustness throughout a trajectory. Test results under multiple parameter configurations are also presented. At the end of this chapter, a VO algorithm is recommended for further development on a fixed-wing SUAS.

4.1 Simulation Results

This section describes and analyzes the results of running the VO algorithms on simulated trajectories generated through SITL in the autonomy simulator.

4.1.1 Default Parameters at 10 m/s.

An initial test is conducted to verify the functionality of the VO algorithms at a lower speed of 10 m/s. The default virtual camera frame rate of 31 fps is used. The maximum roll rate and altitude are varied as specified in Chapter 3. Each VO algorithm is run on the same trajectory three times and the results of the run with the lowest position error is reported. Using the specified camera frame rate, DSO is able to run at 31 Hz, ORB-SLAM2 at 20 Hz, and SVO at 31 Hz.

Figure 29 shows the VO performance of each trajectory at maximum roll rates. The robustness plots in the top row show the percentage of the total trajectory time that the algorithms are able to successfully track and maintain localization. The RMSE plots in the bottom two rows show the position and rotation errors at each altitude and roll rate. The errors are reported as zero if the VO algorithms are unable to successfully initialize for those trajectories and this can be verified through the robustness plots. Table 1 shows the position and error values from the plots in

Figure 29 and highlights the best performing algorithm for each trajectory.

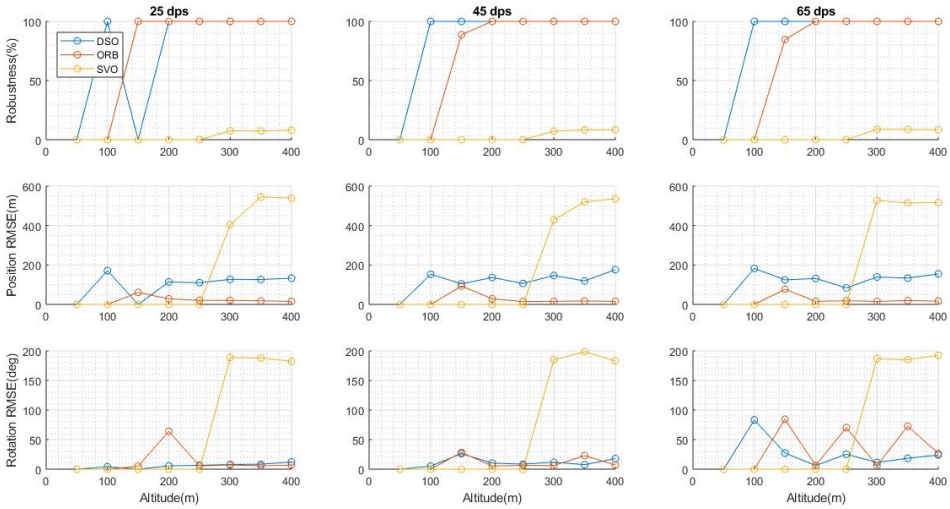

Figure 29. Simulation VO Performance with Default Parameters at 10 m/s. The top row of plots shows the percentage of the total trajectory time that the algorithm is able to maintain tracking. The middle row shows the position errors. The bottom row shows the rotation errors. The columns of plots are organized by roll rates. The individual plot values are measured at mission altitudes.

The results in Figure 29 show that most of the algorithms have difficulty initializing at altitudes below 200 m. This is due to the limited resolution of the satellite imagery making up the terrain. This causes difficulties in picking up enough feature points to initialize the algorithms at lower altitudes and a limited field of view. SVO encounters the most difficulty initializing in the virtual environment and is unable to successfully initialize until 300 m in altitude. SVO also proves to be the most fragile algorithm and experiences difficulty maintaining localization during rotations. This is due to its method of keyframe selection. SVO creates new keyframes strictly based on the Euclidean distance between the newest frame and the previous keyframes, failing to take rotations into account [3]. DSO and ORB-SLAM2 are able to successfully track the entire trajectory for the majority of the test cases. ORB-SLAM2 provides the most accurate position estimates throughout all trajectories in which it was able to initialize. Although ORB-SLAM2 and DSO produce comparable rotation

Table 1. Simulation VO Accuracy with Default Parameters at 10 m/s. The position and rotation errors for trajectories in which a VO algorithm failed to initialize are marked with an X. The algorithm with the lowest position or rotation error for a trajectory is highlighted.

Roll Rate (deg/s)	Altitude (m)	Position RMSE (m)			Rotation RMSE (m)		
		DSO	ORB	SVO	DSO	ORB	SVO
25	50	X	X	X	X	X	X
	100	170.997	X	X	4.075	X	X
	150	X	60.898	X	X	4.914	X
	200	113.955	28.420	X	5.541	64.059	X
	250	110.229	20.434	X	6.504	5.470	X
	300	127.009	20.137	403.935	7.965	7.113	188.684
	350	126.387	18.376	545.184	8.274	5.717	187.95
	400	132.861	15.331	539.435	12.257	6.784	182.351
45	50	X	X	X	X	X	X
	100	152.962	X	X	5.101	X	X
	150	104.605	93.130	X	26.227	28.169	X
	200	136.505	28.474	X	10.358	5.355	X
	250	106.315	14.406	X	8.429	6.415	X
	300	146.665	15.811	428.557	11.835	5.923	184.938
	350	119.18	17.349	520.012	7.558	22.937	198.616
	400	176.371	15.749	535.812	17.827	6.922	183.166
65	50	X	X	X	X	X	X
	100	182.635	X	X	83.158	X	X
	150	124.558	77.100	X	27.253	84.419	X
	200	131.615	15.688	X	6.216	6.455	X
	250	83.809	18.574	X	25.237	70.235	X
	300	138.955	15.458	527.49	11.463	6.142	186.732
	350	133.632	19.362	514.676	18.412	72.723	185.18
	400	154.969	16.639	516.447	23.975	26.571	192.534

estimates with ORB-SLAM2 being slightly more accurate at lower roll rates, DSO provides a more consistent and better estimate at the highest roll rate of 65 deg/s.

Figures 30 and 31 show a representative trajectory from this dataset along with its individual pose data to illustrate the VO performances. This data is obtained from the SUAS flying at 300 m in altitude with a maximum roll rate of 25 deg/s. SVO fails 23 seconds into the trajectory before entering the first turn. DSO drifts further from the truth's position than ORB-SLAM2 as the trajectory progresses while both

maintain close approximations of the rotation.

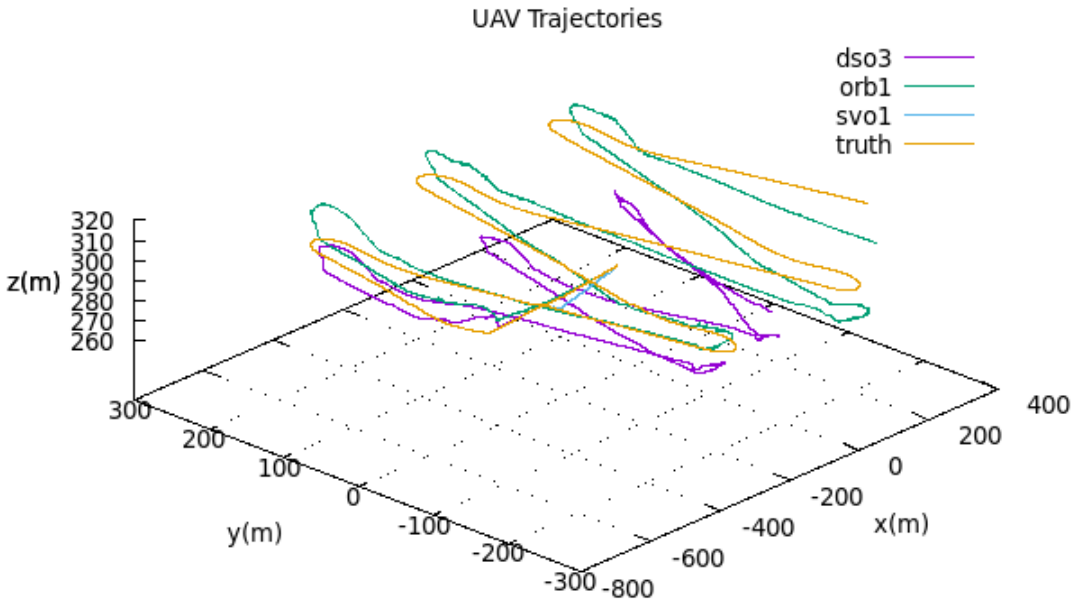

Figure 30. Simulation Trajectories at 10 m/s, 25 deg/s, 300 m

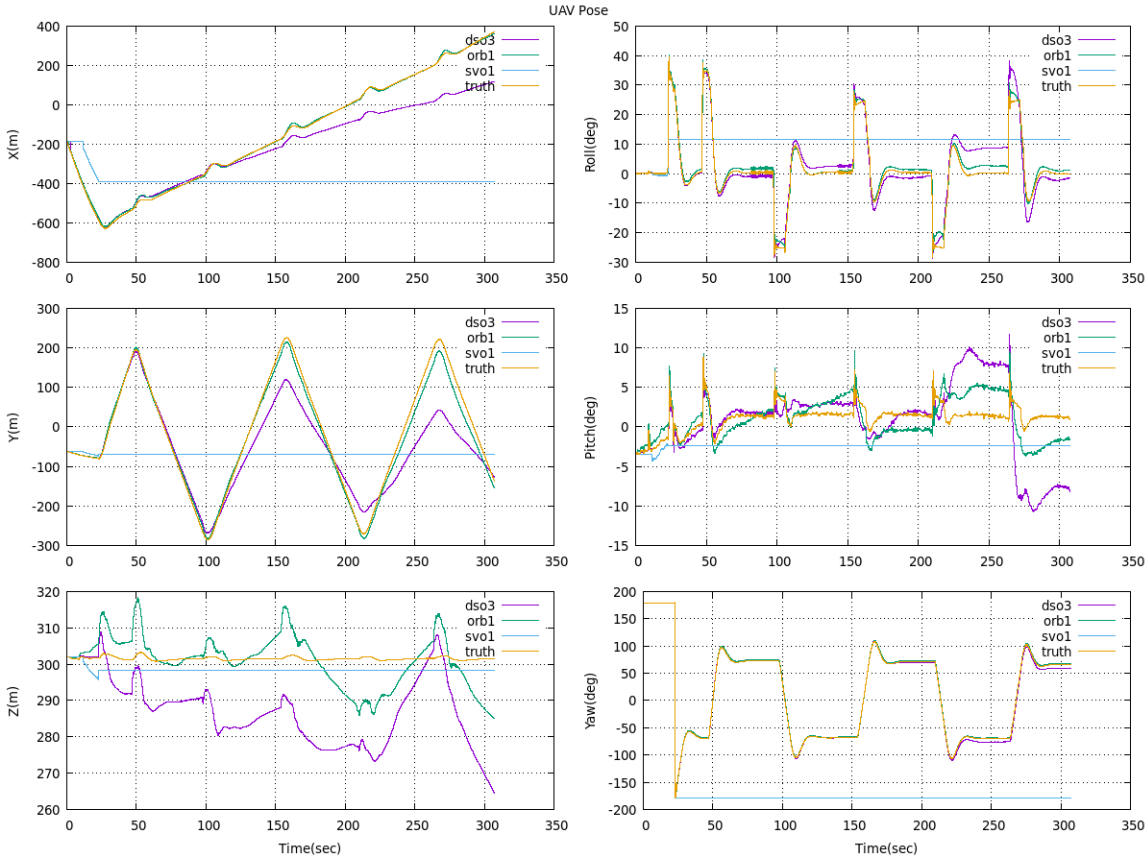

Figure 31. Simulation Pose Data at 10 m/s, 25 deg/s, 300 m. Note that the scales are different for each plot. For example, the scale of the z position plot is much smaller than that of x or y, greatly emphasizing the difference in values.

4.1.2 Default Parameters at 25 m/s.

This baseline test establishes the VO performances with the same default parameters and environment configurations as the previous test with the exception of the SUAS airspeed. This is set to 25 m/s and will remain the operational airspeed for the remaining simulation tests. Figure 32 and Table 2 show the VO performance results under these conditions.

Although the algorithms are able to maintain high levels of robustness at a maximum roll rate of 25 deg/s, the operating limit is at 45 deg/s. ORB-SLAM2 is able to

maintain localization throughout most of the trajectories at this roll rate while DSO experiences frequent failures and SVO continues to fail on the first turns. All algorithms have difficulty maintaining localization at roll rates of 65 deg/s. ORB-SLAM2 continues to maintain a more accurate position estimate than DSO. DSO exhibits a slightly more accurate rotation estimate at lower roll rates of 25 deg/s although this is reversed at 45 deg/s with ORB-SLAM2 providing better estimates.

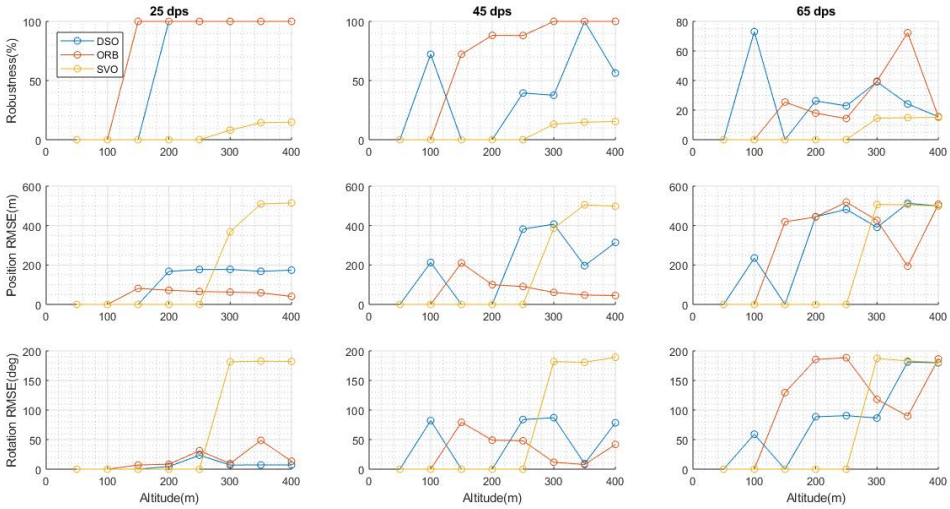

Figure 32. Simulation VO Performance with Default Parameters at 25 m/s. The top row of plots shows the percentage of the total trajectory time that the algorithm is able to maintain tracking. The middle row shows the position errors. The bottom row shows the rotation errors. The columns of plots are organized by roll rates. The individual plot values are measured at mission altitudes.

4.1.3 Framerate at 5 fps.

This test uses the default parameters and an airspeed of 25 m/s but throttles the frame rate of the virtual camera to 5 fps. Figure 33 shows the VO performance results. All of the algorithms suffer a decrease in both robustness and pose accuracy from the lower frame rate. The algorithms are less capable of maintaining localiza-

Table 2. Simulation VO Accuracy with Default Parameters at 25 m/s. The position and rotation errors for trajectories in which a VO algorithm failed to initialize are marked with an X. The algorithm with the lowest position or rotation error for a trajectory is highlighted.

Roll Rate (deg/s)	Altitude (m)	Position RMSE (m)			Rotation RMSE (m)		
		DSO	ORB	SVO	DSO	ORB	SVO
25	50	X	X	X	X	X	X
	100	X	X	X	X	X	X
	150	X	81.206	X	X	7.047	X
	200	167.809	71.906	X	4.535	8.326	X
	250	176.991	65.365	X	23.527	31.138	X
	300	177.575	62.717	368.72	7.012	9.573	181.637
	350	167.853	58.907	509.857	7.111	48.707	182.546
	400	173.911	40.369	514.787	7.047	13.425	182.213
45	50	X	X	X	X	X	X
	100	212.844	X	X	82.186	X	X
	150	X	210.371	X	X	79.259	X
	200	X	100.056	X	X	49.045	X
	250	381.837	90.671	X	83.847	48.052	X
	300	407.325	61.127	386.2	87.066	11.731	182
	350	196.225	47.466	505.137	9.248	7.815	180.623
	400	313.935	45.072	497.773	78.373	41.819	189.143
65	50	X	X	X	X	X	X
	100	235.459	X	X	59.036	X	X
	150	X	419.035	X	X	129.506	X
	200	444.963	443.574	X	88.568	185.526	X
	250	482.253	519.21	X	90.544	188.444	X
	300	390.829	426.205	506.941	86.453	118.167	187.349
	350	513.123	194.008	506.12	181.218	89.653	182.838
	400	499.621	508.358	498.452	179.734	186.319	180.706

tion at higher roll rates and during aggressive maneuvers. SVO and DSO are most impacted by the lower frame rate since they rely on the Gauss-Newton optimization for image alignment and pose estimation, leading to a heavy reliance on higher frame rates for robust calculations [4] [21]. They also do not conduct outlier rejection using approaches like RANSAC, so they depend on small movements of the features tracked on the image frame. ORB-SLAM2 remains the most robust algorithm but also experiences higher errors and failure rates at increased roll rates. The results confirm

that all of the VO algorithms benefit in accuracy from higher frame rates and also that aggressive maneuvers necessitate a high frame rate to maintain localization.

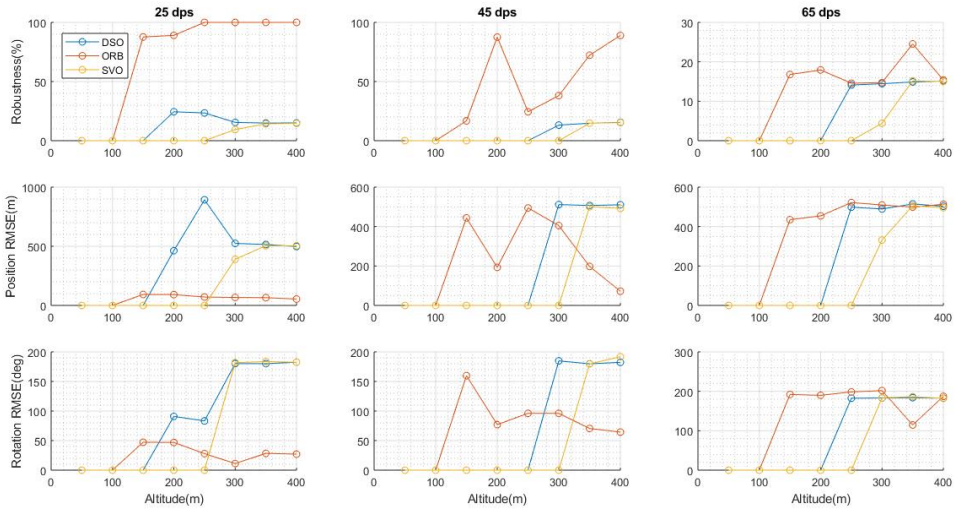

Figure 33. Simulation VO Performance at 5 fps. The top row of plots shows the percentage of the total trajectory time that the algorithm is able to maintain tracking. The middle row shows the position errors. The bottom row shows the rotation errors. The columns of plots are organized by roll rates. The individual plot values are measured at mission altitudes.

4.1.4 Lowered VO thresholds.

Figures 34, 35, and 36 show the results of lowering VO pixel selection thresholds as described in Section 3.3.4. DSO's image gradient threshold is lowered to five. ORB-SLAM2 and SVO's FAST score thresholds are lowered to two. In all cases, lowering the thresholds allow the algorithms to track at lower altitudes. However, this comes with a slight cost in accuracy at higher altitudes since lower quality pixels or features are used to calculate pose estimates. Performances are mixed at higher roll rates and altitudes where the VO algorithms have difficulty maintaining localization. SVO is still unable to track past the first turn in the trajectory.

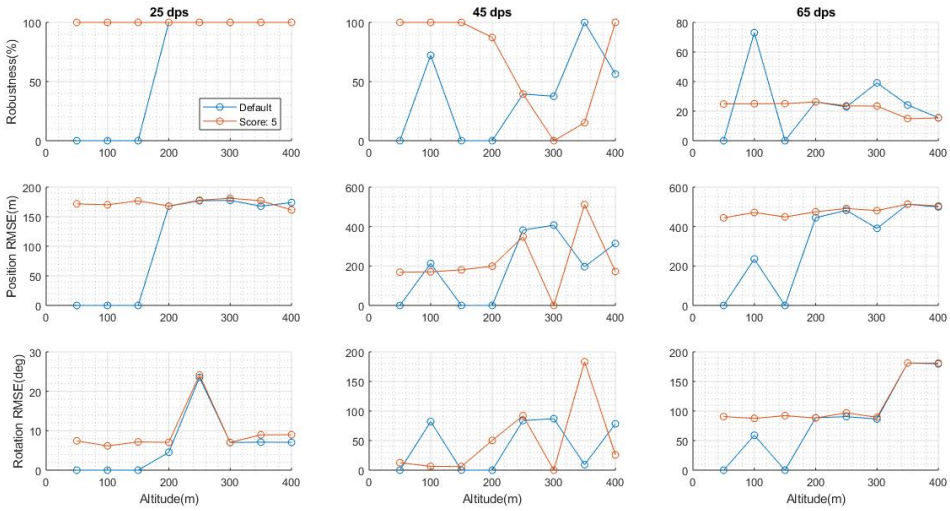

Figure 34. Simulation DSO Image Gradient Threshold Performance. The graphs compare the performance of DSO using default configurations against lowering the image gradient threshold to five. The top row of plots shows the percentage of the total trajectory time that the algorithm is able to maintain tracking. The middle row shows the position errors. The bottom row shows the rotation errors. The columns of plots are organized by roll rates. The individual plot values are measured at mission altitudes.

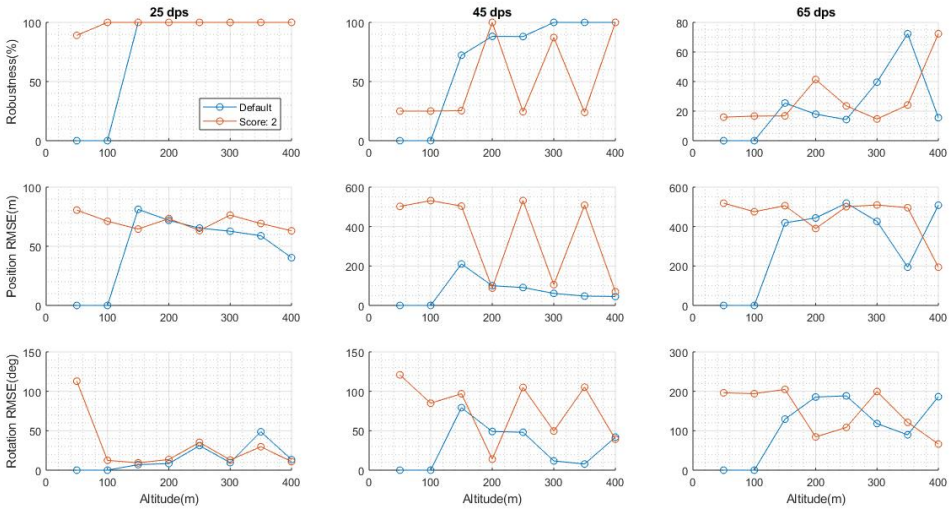

Figure 35. Simulation ORB-SLAM2 FAST Threshold Performance. The graphs compare the performance of ORB-SLAM2 using default configuration against lowering the FAST threshold to two.

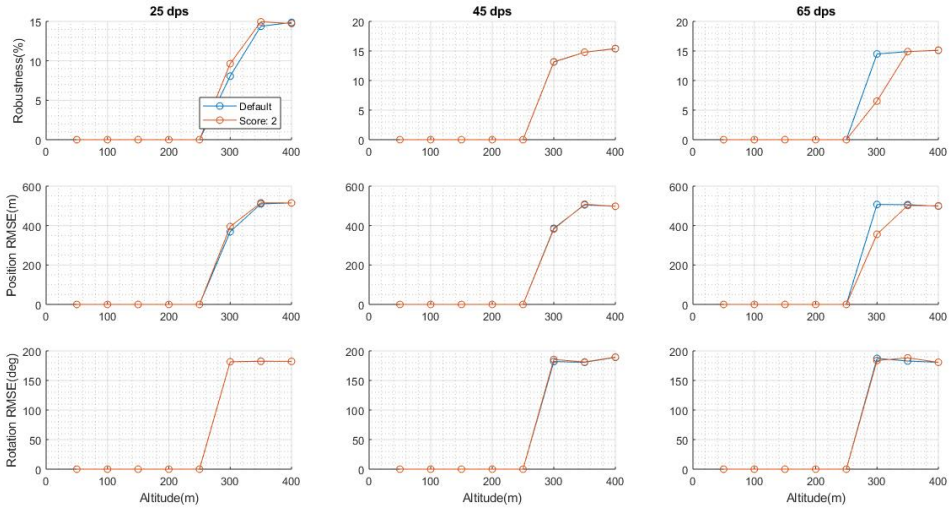

Figure 36. Simulation SVO FAST Threshold Performance. The graphs compare the performance of SVO using default configurations against lowering the FAST threshold to two.

4.1.5 SVO Keyframe Selection Distance.

SVO's keyframe selection distance is lowered to 5% to induce a higher frequency of keyframe selection and increase robustness for fast motions. However, Figure 37 shows that this is not enough to overcome a rolling motion even at 25 deg/s. Figure 38 shows a representative dataset from a trajectory obtained at a roll rate of 25 deg/s and an altitude of 350 m in which both SVO runs with default parameters and a reduced keyframe selection distance fail at the first turn. This is due to the fact that SVO's keyframe selection criteria is strictly based on Euclidean distance and fails to account for changes in the visual field that may come with rotations [3].

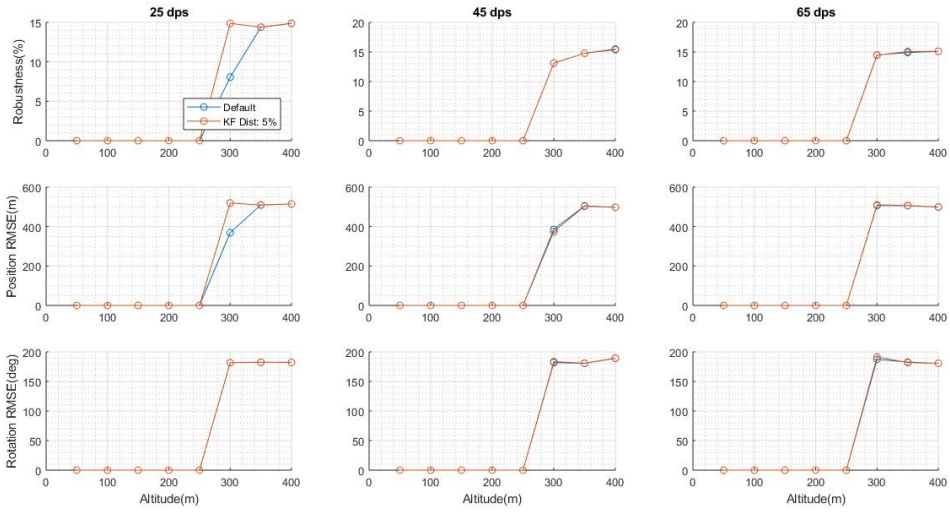

Figure 37. Simulation SVO Keyframe Distance Performance. The graphs compare the performance of SVO using default configurations against lowering the keyframe selection distance to 5%. The top row of plots shows the percentage of the total trajectory time that the algorithm is able to maintain tracking. The middle row shows the position errors. The bottom row shows the rotation errors. The columns of plots are organized by roll rates. The individual plot values are measured at mission altitudes.

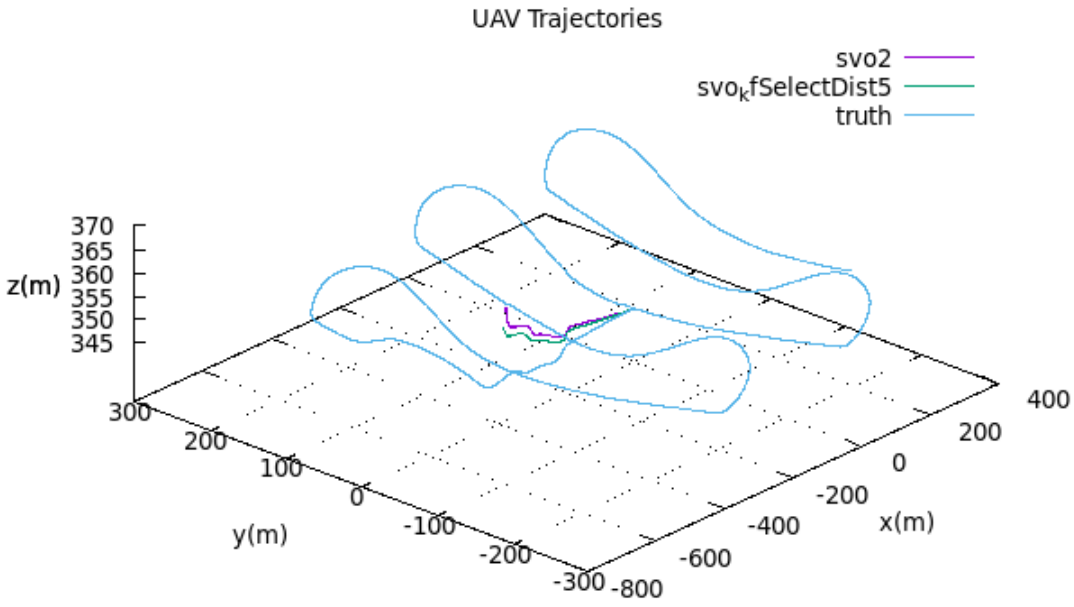

Figure 38. Simulation SVO Trajectories at 25 m/s, 25 deg/s, 350 m

4.2 Real-World Flight Test Results

The data collected from real-world flight tests over Camp Atterbury contain several problems that raise difficulties in testing the VO algorithms. The first major problem is inaccuracies in the truth data, especially in the reported yaw angle of the SUAS. Although the autopilot reports filtered attitude values with GPS that are accurate within five degrees, this was not captured in the datasets. Instead, only the GPS-denied attitude data was captured and the yaw angle from this data repeatedly drifts 90 degrees or more throughout all of the flights. The second problem involves the limitations in the exposure time set on the camera to achieve its maximum frame rate of 33 fps. This leads to a saturated image on bright days which, when combined with the low texture environment of the airfield, provides a difficult challenge for the VO algorithms to be able to detect and track features or pixels. An example image from the third flight exhibiting these characteristics is shown in Figure 39. For this

reason, none of the algorithms are able to initialize and track using the raw imagery from the first three flights. The algorithms are able to track parts of the fourth flight but no algorithm is able to track throughout the entire trajectory on this flight either.

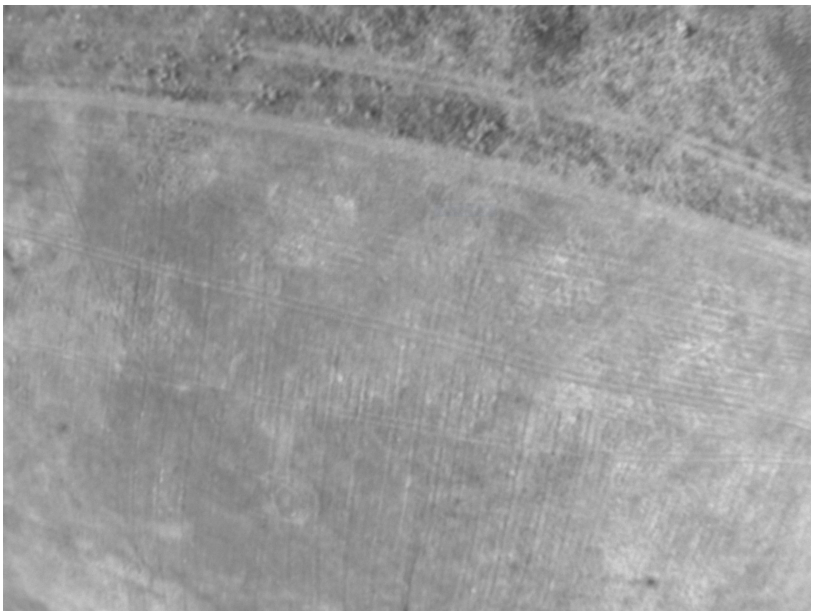

Figure 39. Saturated Low Texture Image

The results reported in this section is from a 9.6 km section of the fourth flight in which the SUAS flies in a grid pattern at 250 m altitude as shown by Figure 40. The grid starts from the west and performs three counter-clockwise loops moving east. The flight time for this trajectory is 284 s.

Figure 40. Real-World Truth Trajectory

DSO is unable to successfully initialize using either default parameters or by lowering the image gradient threshold to five or two. ORB-SLAM2 with the default parameters is the most successful in tracking throughout the entire trajectory. Figure 41 illustrates ORB-SLAM2's estimated trajectory against the unfiltered truth trajectory data. Figure 42 shows the detailed pose estimates. The most extreme errors occur in the z axis as the algorithm initially miscalculates the pitch, leading to oscillations in altitude between the North and South ends of the trajectory. Figure 43 shows the accumulating drift in the position error throughout the trajectory. For this trajectory, the total position RMSE is 321 meters and the rotation RMSE is 112 degrees. Additional tests conducted by lowering ORB-SLAM2's FAST threshold to values between six and two result in the algorithm being unable to initialize throughout the trajectory.

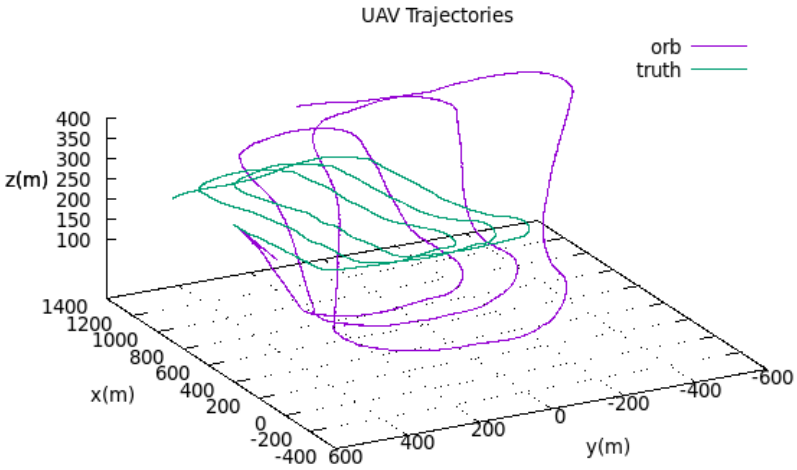

Figure 41. Real-World ORB-SLAM2 Trajectory

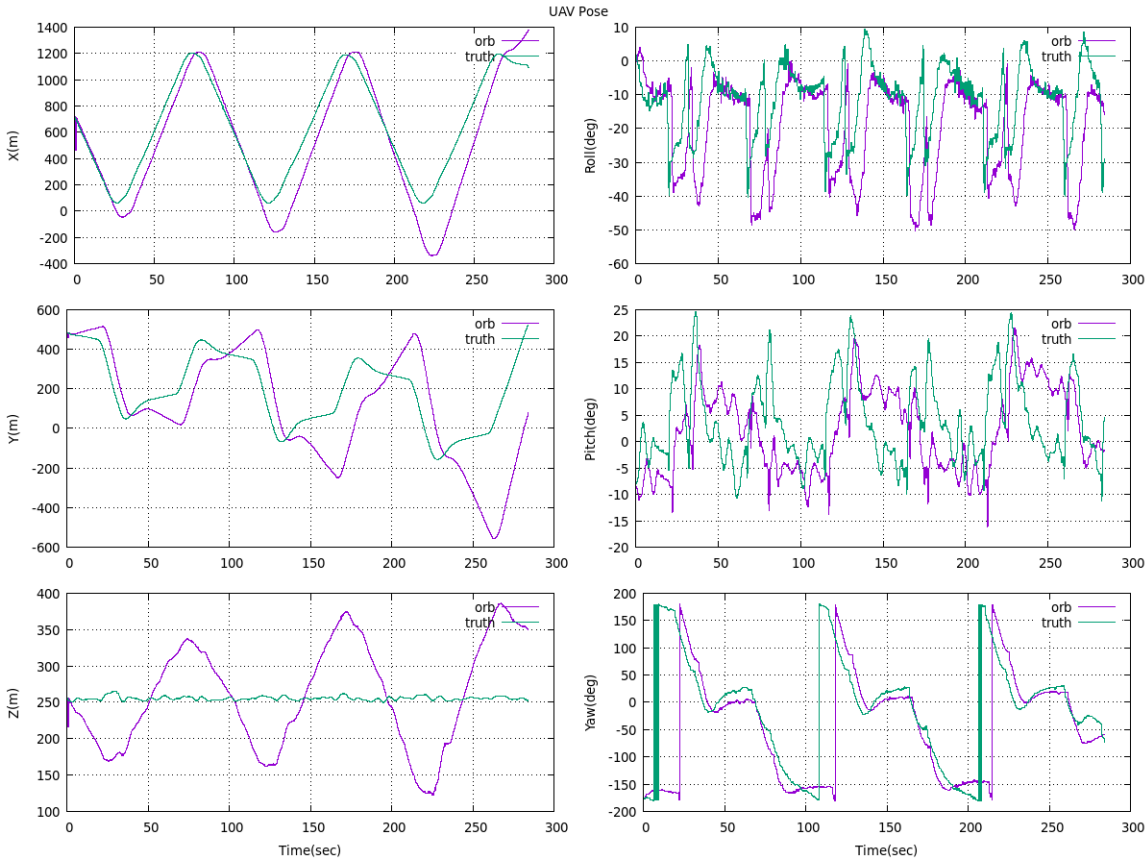

Figure 42. Real-World ORB-SLAM2 Pose

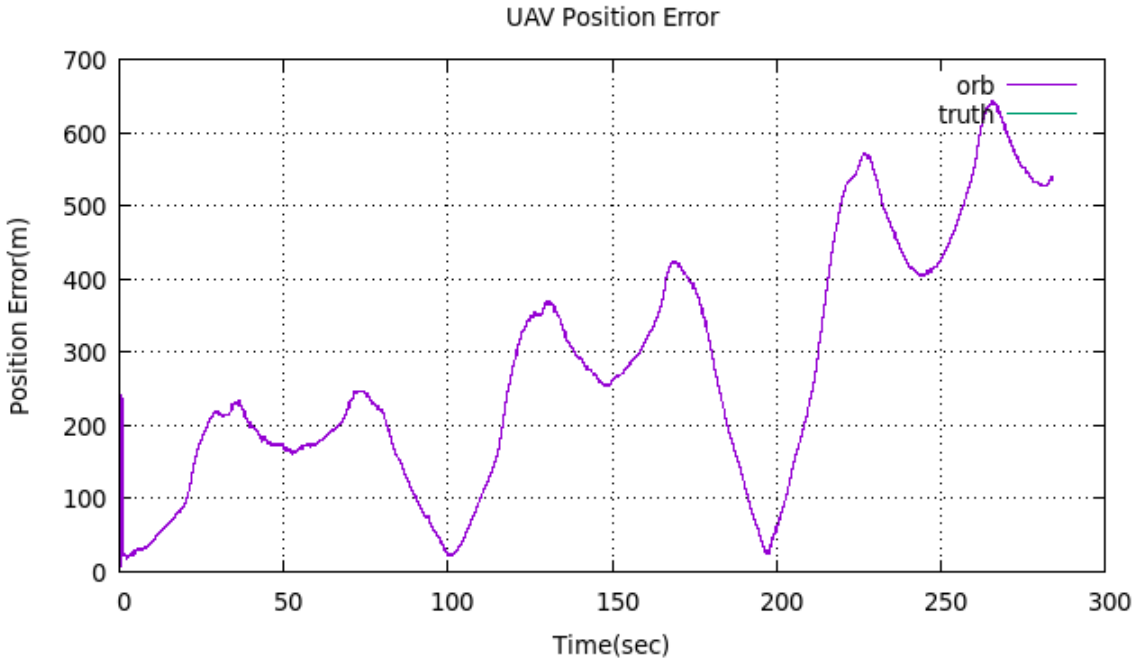

Figure 43. Real-World ORB-SLAM2 Position Error

For comparison, ORB-SLAM2 was also run on virtual imagery from the SUAS flying the unfiltered real-world trajectory dataset in the virtual environment. Both trajectories estimated from the virtual and real-world imagery are shown in Figure 44 and their pose data is shown on Figure 45. The algorithm is able to obtain better initialization using the virtual imagery since the lighting conditions are constant and easier to detect features in. However, it loses localization after the second loop due to erratic jumps in the unfiltered trajectory data. In order to draw better comparisons of algorithm performance in the virtual and real world, ORB-SLAM2 should be run on trajectory data that is filtered and gathered at higher frequencies.

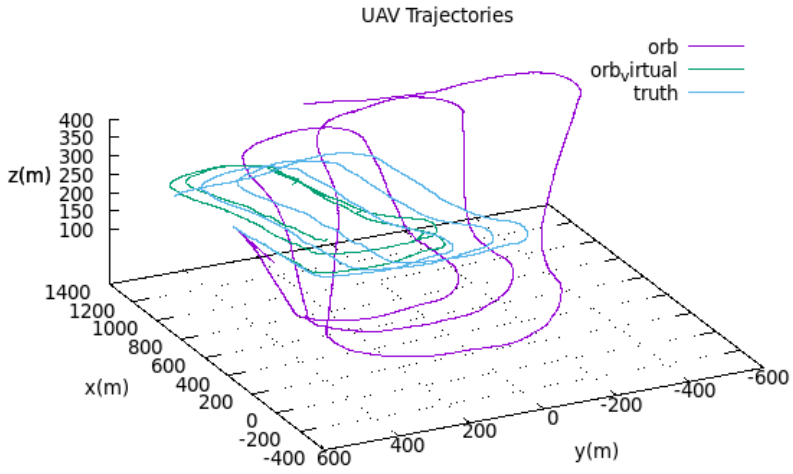

Figure 44. ORB-SLAM2 Trajectory from Real-World Trajectory and Virtual Imagery

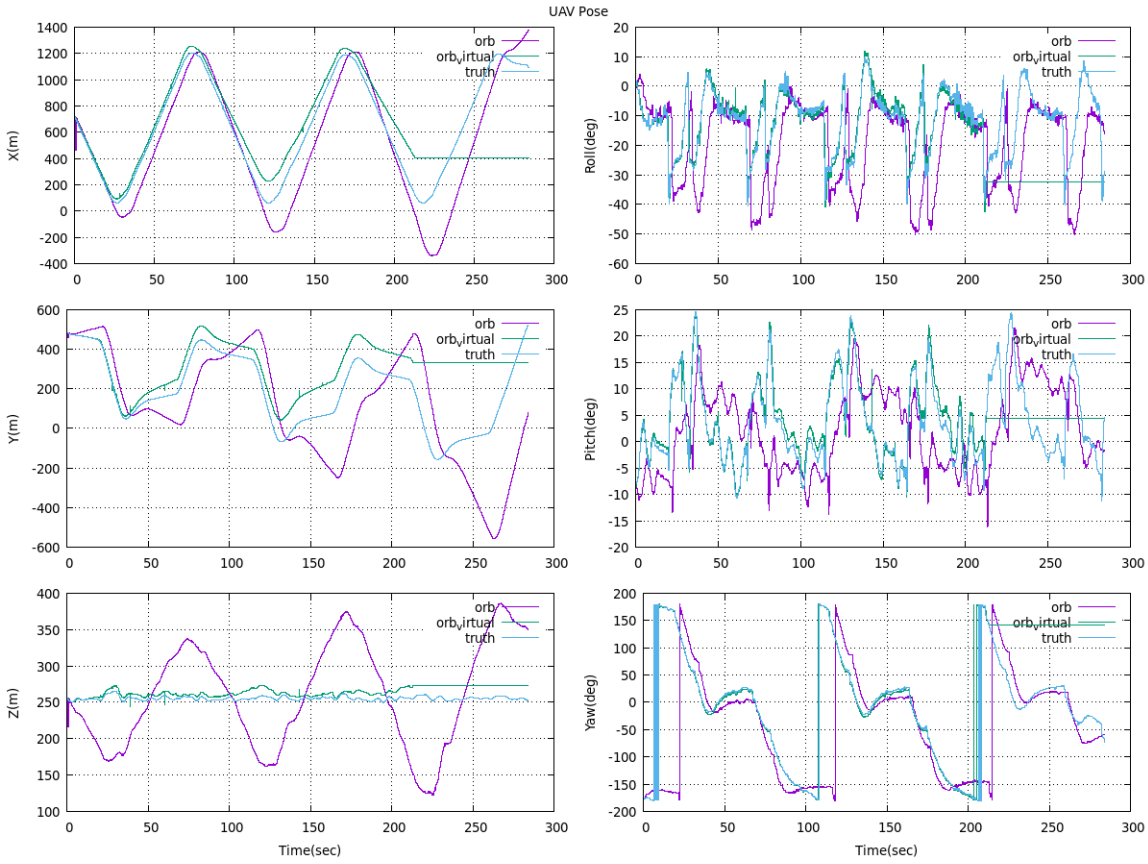

Figure 45. ORB-SLAM2 Pose Data from Real-World Trajectory and Virtual Imagery

SVO experiences the same difficulties handling rotations as encountered during simulation. The following combinations of parameters are tested:

1. Default parameters
2. Reduced keyframe selection distance: 5%
3. Reduced FAST score threshold: 5
4. Reduced keyframe selection distance to 5% and FAST score threshold to 5

As shown in Figure 46, although all cases are able initialize and track through the first leg, only the combination of reducing both the keyframe selection distance and the FAST score threshold allows the algorithm to continue tracking past the first turn. However, it shortly loses localization afterwards due to the difficult texture of the airfield in the image.

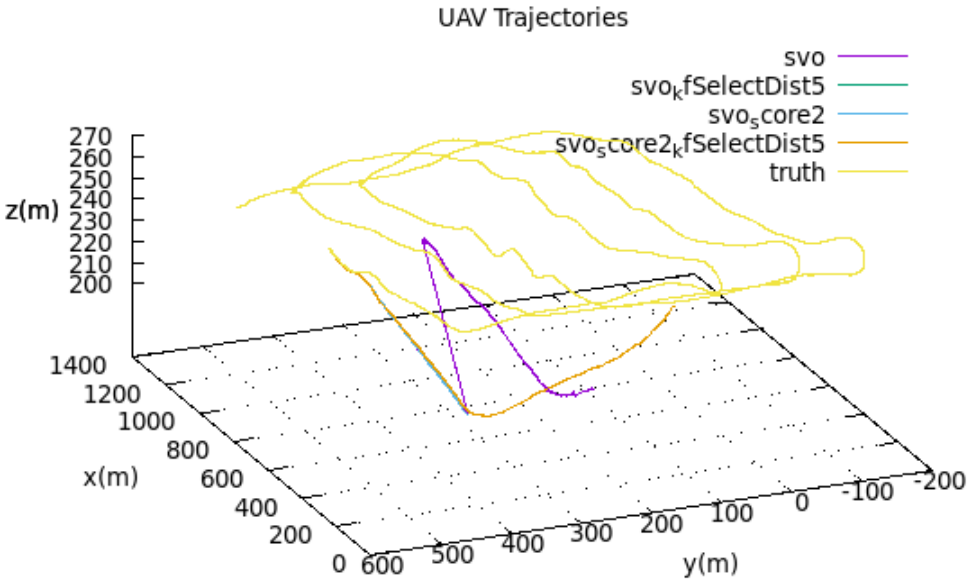

Figure 46. Real-World SVO Trajectory

4.3 Analysis

Overall, ORB-SLAM2 proves to be the most flexible VO algorithm that is suitable for fixed-wing applications. Both simulation and real-world flight tests show a robust initializer and tracking capability. Although it is not able to achieve the frame rates of SVO or DSO, ORB-SLAM2 is able to track far more robustly at higher speeds and sharper turns, where the combination of 25 m/s and 45 deg/s is the operational limit. Its robustness is further highlighted by its capability to maintain localization under a greatly throttled image stream rate. ORB-SLAM2 also has the distinct advantage in further improvement since a SLAM scheme is incorporated naturally in the design of the algorithm.

SVO is fast but proves to be very fragile, especially during rotations. SVO is unable to cope with any of the turns during simulation or real-world flight tests, even at the slowest speeds and roll rates. This is primarily due to its method of designating new keyframes by a measure of distance. If using SVO for fixed-wing applications, this heuristic must be modified to another method such as that of ORB-SLAM2 or DSO where keyframe creation is based on the measure of changes in its field of view.

DSO's greatest weakness lies in the initializer. The authors note on their git repository[1] that the current initializer is not very good and requires slow, easy movements with lots of translation and little rotation. This is much easier to achieve in simulation with perfectly smooth flights creating ideal environments for initialization. However, real-world flights prove to be much more difficult as the SUAS invariably encounters turbulence from wind. This is shown by DSO being unable to successfully initialize in any of the real-world datasets. If using DSO, the initializer should be replaced by a faster method such as through feature-based homography or RANSAC.

[1] J. Engel, 'Direct Sparse Odometry', 2018. [Online]. Available: https://github.com/JakobEngel/dso. [Accessed: 23- Jan- 2019].

The initializer could also be either augmented or completely replaced by IMU measurements. Another drawback to DSO is that it is not as amenable to incorporating loop closures as other feature-based VO algorithms, since many of the current loop closure frameworks make use of features for the bag-of-words approach [65].

The imagery data collected from the real-world flight tests should be corrected for brightness and re-run to test whether the VO algorithms are able to process the data. Additionally, the truth trajectory data from these flight tests currently contain large inaccuracies in the attitude data. This must be corrected by either filtering the sensor data or repeating the flight tests to gather data from more sensor sources before also filtering them.

4.4 Summary

This chapter provided the experimentation results from simulation and real-world flight tests conducted over Camp Atterbury. In simulation, each of the VO algorithms were tested with default parameters at 10 m/s and 25 m/s. The effects of throttling the frame rate were also explored. ORB-SLAM2 proved to be the most robust and accurate in the majority of cases, particularly at higher speeds and roll rates. For both the simulation and real-world data, the VO pixel gradient and feature thresholds were lowered and the keyframe selection distance in SVO was also reduced. Again, ORB-SLAM2 proved to be the most robust algorithm, especially on the real-world dataset. A further analysis was provided recommending ORB-SLAM2 for further development on a fixed-wing SUAS and highlighting improvements to be made on DSO and SVO before applying these algorithms on fixed-wing SUAS.

V. Conclusion

This chapter provides a summary of this research. The experiment and its results are reviewed along with key conclusions drawn from this work. Finally, recommendations for future work are presented.

5.1 Experiment

Three of the current state-of-the-art VO algorithms DSO, SVO, and ORB-SLAM2 were applied to fixed-wing flight and performances were compared using both simulation and real-world flight tests over Camp Atterbury, Indiana. Simulation tests were conducted to analyze the performance of the VO algorithms under varying airspeeds of 10 and 25 m/s, maximum roll rates of 25, 45, and 65 deg/s and mission altitudes from 50-400 m in 50 m increments over a 3.3 km trajectory. The VO algorithms were also tested on real-world datasets and results were presented for a 9.6 km trajectory where the aircraft flew at 20-30 m/s at a 250 m altitude.

ORB-SLAM2 was found to be the most robust algorithm and is recommended for further development in fixed-wing applications. ORB-SLAM2 was able to continue maintaining localization at higher roll rates and provided more accurate estimates than DSO. DSO's initializer proved to be a problem for the real-world dataset and failed to successfully initialize throughout the trajectory. SVO was unable to handle sharp rotations and repeatedly failed on turns. The operational limits of ORB-SLAM2 were determined to be an airspeed of 25 m/s and a roll rate of 45 deg/s.

5.2 Future Work

Further analysis on ORB-SLAM2 can be conducted to determine the additional accuracy afforded by enabling its loop closure capabilities in both simulation and

real-world flights. It should then be integrated with a filter to augment it with measurements from additional sensors such as an IMU. This will allow for greater accuracy as well as robustness under fast movements where ORB-SLAM2 by itself would normally fail. This visual-inertial SLAM system will also allow continuous computation of the absolute scale. This navigation solution should be tested on an onboard CPU to ensure that real-time processing is a viable solution on a compact system.

Another line of effort can involve comparing VO results between the virtual and real worlds. This can be done by using real-world imagery and simulated imagery from real-world trajectory data. However, since the trajectory data gathered in this work contains too many inaccuracies in attitude, the truth data needs to be filtered and corrected or new trajectory data must be gathered to investigate this topic. Determining this accuracy would allow for greater insight into the validity and limitations of the simulated VO results.

Finally, a novel work could involve using the virtual environment to train deep learning VO algorithms. The greatest obstacle in deep learning VO is that it must first be trained through flights and reinforced with knowledge of the truth data. Gathering accurate 6-DOF truth data at high-frequencies is a difficult task, especially on a SUAS. Also, conducting multiple flight tests over a variety of target environments may be an expensive, time-consuming effort or not feasible at all. However, the virtual environment addresses both of these issues, giving easy access to perfect truth data at 31 Hz and flight tests over any environment with publicly available elevation data and satellite imagery. This work could investigate the use of the virtual environment and simulated flight data to train the VO network before deploying it in the real world.

Bibliography

1. Deputy Chief of Staff for ISR (A2), "Small Unmanned Aircraft Systems (SUAS) Flight Plan: 2016 - 2036," p. 94, 2016.

2. D. Scaramuzza and F. Fraundorfer, "Visual Odometry," *IEEE Robotics & Automation Magazine*, no. December, 2011.

3. C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast Semi-Direct Monocular Visual Odometry," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15–22, 2014.

4. J. Engel, V. Koltun, and D. Cremers, "Direct Sparse Odometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, pp. 611–625, 2018.

5. R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

6. S. Nykl, C. Mourning, N. Ghandi, and D. Chelberg, ""A Flight Tested Wake Turbulence Aware Altimeter"," *Advances in Visual Computing*, pp. 219–228, 2011.

7. G. Balamurugan, J. Valarmathi, and V. P. Naidu, "Survey on UAV navigation in GPS denied environments," *International Conference on Signal Processing, Communication, Power and Embedded System, SCOPES 2016 - Proceedings*, pp. 198–204, 2017.

8. S. Thrun, M. Montemerlo, and A. Aron, "Probabilistic Terrain Analysis For High-Speed Desert Driving," *Robotics: Science and Systems II*, 2006.

9. A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann, "6D SLAM - 3D mapping outdoor environments," *Journal of Field Robotics*, vol. 24, no. 8-9, pp. 699–722, 2007.

10. A. I. Comport, E. Malis, and P. Rives, "Accurate quadrifocal tracking for robust 3D visual odometry," *Proceedings - IEEE International Conference on Robotics and Automation*, no. April, pp. 40–45, 2007.

11. K. Schmid, T. Tomic, F. Ruess, H. Hirschmuller, and M. Suppa, "Stereo vision based indoor/outdoor navigation for flying robots," *IEEE International Conference on Intelligent Robots and Systems*, pp. 3955–3962, 2013.

12. R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," *2011 10th IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2011*, pp. 127–136, 2011.

13. C. Kerl and D. Cremers, "Robust Odometry Estimation for RGB-D Cameras," *IEEE International Conference on Robotics and Automation*, pp. 3748–3754, 2013.

14. D. Gutierrez-Gomez, W. Mayol-Cuevas, and J. J. Guerrero, "Dense RGB-D visual odometry using inverse depth," *Robotics and Autonomous Systems*, vol. 75, pp. 571–583, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2015.09.026

15. K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, "An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics," *Intelligent Industrial Systems*, vol. 1, no. 4, pp. 289–311, 2015.

16. B. Y. H. Durrant-whyte, T. I. M. Bailey, P. Cheeseman, J. Crowley, and H. Durrant, "Simultaneous Localization and Mapping : Part I," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

17. R. Hartley and R. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge U.K: Cambridge Univ. Press, 2004.

18. Opencv: Camera calibration. [Online]. Available: https://docs.opencv.org/3.1.0/dc/dbb/tutorial_py_calibration.html

19. R. C. Leishman, T. W. Mclain, R. W. Beard, R. C. Leishman, T. W. Mclain, and R. W. Beard, "Relative Navigation Approach for Vision-Based Aerial GPS-Denied Navigation," *J Intell Robot Syst*, vol. 74, pp. 97–111, 2014.

20. D. O. Wheeler and D. P. Koch, "Relative Navigation : A Keyframe-Based Approach for Observable GPS-Degraded Navigation," *IEEE Controls Systems Magazine*, vol. 38, no. 4, pp. 30–48, 2018.

21. C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, "Semi-Direct Visual Odometry for Monocular , Wide-angle, and Muti-Camera Systems," *IEEE Transactions on Robotics*, vol. 33, pp. 249–265, 2017.

22. R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM : Dense Tracking and Mapping in Real-Time," *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

23. G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," *IEEE and ACM International Symposium on Mixed and Augmented Reality*, p. 1–10, 2007.

24. J. Engel, T. Sch, and D. Cremers, "LSD-SLAM: Large-scale Direct Monocular SLAM," in *European Conference on Computer Vision*, 2014, pp. 834–849.

25. M. Hassaballah, A. A. Abdelmgeid, and H. A. Alshazly, *Image Feature Detectors and Descriptors*, 2016, vol. 630.

26. D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, 2004.

27. H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," *European Conference on Computer Vision*, 2006.

28. E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," *European Conference on Computer Vision*, vol. 1, 2006. [Online]. Available: papers://9cdf6d24-b953-4ed9-8d83-e25ff0393b0d/Paper/p3567

29. E. Rublee, V. Rabaud, and K. Konolige, "ORB : An Efficient Alternative to SIFT or SURF," *Intl. Conf. Computer Vision*, pp. 1–5, 2011.

30. M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Application to Image Analysis and Automated Cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

31. M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor Micro Aerial Vehicle," *J. Field Robotics*, vol. 23, no. 0, p. 1–20, 2015.

32. D. Carson, "Aerial Visual-Inertial Odometry Performance Evaluation," Master's thesis, Air Force Institute of Technology, 2017.

33. G. Ellingson, K. Brink, and T. McLain, "Relative visual-inertial odometry for fixed-wing aircraft in GPS-denied environments," *2018 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pp. 786–792, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8373454/

34. N. Sünderhauf, K. Konolige, S. Lacroix, and P. Protzel, "Visual Odometry Using Sparse Bundle Adjustment on an Autonomous Outdoor Vehicle," *Autonome Mobile Systeme 2005*, pp. 157–163, 2005. [Online]. Available: http://link.springer.com/10.1007/3-540-30292-1_20

35. F. Fraundorfer, D. Scaramuzza, and M. Pollefeys, "A constricted bundle adjustment parameterization for relative scale estimation in visual odometry," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1899–1904, 2010.

36. D. Scaramuzza and F. Fraundorfer, "Visual Odometry Part II," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 80–92, 2011.

37. M. Irani and P. Anandan, "All About Direct Methods," *ICCV workshop on Vision Algorithms*, pp. 267–277, 1999.

38. H. P. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover." *Stanford Univ Ca Dept of Computer Science*, p. 175, 1980.

39. D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, vol. 1, 2004.

40. C. Harris and M. Stephens, "A Combined Corner and Edge Detector," *Alvey Vision Conferenc*, 1988.

41. A. J. Davison, ""MonoSLAM: Real-time single camera SLAM."," *IEEE transactions on pattern analysis and machine intelligence 29.6 (2007).*, vol. 29, no. 6, pp. 1052–1067, 2007.

42. J. Shi and C. Tomasi, "Good Features to Track," *IEEE Conf. Computer Vision and Pattern Recognition*, pp. 593–600, 1994.

43. J. Engel, J. Sturm, and D. Cremers, "Semi-dense visual odometry for a monocular camera," in *Proceedings of the IEEE International Conference on Computer Vision*, Sydney, Australia, 12 2013, pp. 1449–1456.

44. A. Glover, S. Member, W. Maddern, S. Member, M. Warren, S. Reid, M. Milford, and G. Wyeth, "OpenFABMAP - An Open Source Toolbox for Appearance-based Loop Closure Detection," *IEEE International Conference on Robotics and Automation*, pp. 4730–4735, 2012.

45. R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A General Framework for Graph Optimization," *Intl. Conf. on Robotics and Automation (ICRA)*, 2011.

46. J. Engel, V. Usenko, and D. Cremers, "A Photometrically Calibrated Benchmark For Monocular Visual Odometry," *Computing Research Repository (CoRR)*, 2016.

47. L. von Stumberg, V. Usenko, and D. Cremers, "Direct Sparse Visual-Inertial Odometry Using Dynamic Marginalization," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2510–2517, 2018.

48. X. Gao, R. Wang, N. Demmel, and D. Cremers, "LDSO: Direct Sparse Odometry with Loop Closure," *Computing Research Repository (CoRR)*, 2018. [Online]. Available: http://arxiv.org/abs/1808.01111

49. R. Mur-Artal, J. M. Montiel, and J. D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

50. M. Pizzoli, C. Forster, and D. Scaramuzza, "REMODE: Probabilistic, monocular dense reconstruction in real time," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2609–2616, 2014.

51. C. Forster, M. Faessler, F. Fontana, M. Werlberger, and D. Scaramuzza, "Continuous on-board monocular-vision-based elevation mapping applied to autonomous landing of micro aerial vehicles," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2015-June, no. June, pp. 111–118, 2015.

52. H. Wen, S. Wang, R. Clark, and N. Trigoni, "{DeepVO}: Towards End to End Visual Odometry with Deep Recurrent Convolutional Neural Networks," *Proceedings of the {IEEE} International Conference on Robotics and Automation ({ICRA})*, pp. 2043–2050, 2017.

53. R. Li, S. Wang, Z. Long, and D. Gu, "UnDeepVO: Monocular Visual Odometry through Unsupervised Deep Learning," *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

54. A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," *International Journal of Robotics Research (IJRR)*, no. October, pp. 1–6, 2013.

55. J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A Benchmark for the Evaluation of RGB-D SLAM Systems," *Inernational Conference on Intelligent Robot Systems (IROS)*, 2012.

56. M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

57. A. Majdik, C. Till, and D. Scaramuzza, "The Zurich Urban Micro Aerial Vehicle Dataset," *I. J. Robotics Res.*, vol. 36, pp. 269–273, 2017.

58. D. O. Wheeler and D. P. Koch, "Relative Navigation of Autonomous GPS- Degraded Micro Air Vehicles," *IEEE Control Systems Magazine*, vol. 38, no. 4, pp. 30–48, 2018.

59. P. Alcantarilla, J. Nuevo, and A. Bartoli, "Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces," *Procedings of the British Machine Vision Conference 2013*, pp. 1–13, 2013.

60. S. Baker and I. Matthews, "Lucas-Kanade 20 Years On : A Unifying Framework : Part 1 2 Background : Lucas-Kanade," *International Journal of Computer Vision*, vol. 56, no. 3, pp. 221–255, 2004.

61. M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: An Open-Source Robot Operating System," *ICRA Workshop on Open Source Software*, 2009.

62. A. S. Huang, E. Olson, and D. C. Moore, "LCM: Lightweight Communications and Marshalling," *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, no. Lcm, pp. 4057–4062, 2010.

63. S. Nykl, C. Mourning, M. Leitch, D. Chelberg, T. Franklin, and C. Liu, "An Overview of the STEAMiE Educational Game Engine," *Proceedings - Frontiers in Education Conference, FIE*, pp. 21–25, 2008.

64. I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, "A Brief Introduction to OpenCV," *MIPRO, 2012 Proceedings of the 35th International Convention*, pp. 1725–1730, 2012. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6240859

65. M. Cummins and P. Newman, "FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance," *International Journal of Robotics Research*, vol. 27, no. 6, pp. 647–665, 2008.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
3–21–19	Master's Thesis	Sept 2017 — Mar 2019

4. TITLE AND SUBTITLE

Monocular Visual Odometry for Fixed-Wing Small Unmanned Aircraft Systems

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Kim, Kyung M.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering an Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-19-M-036

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RYWN
Bldg 600
WPAFB OH 45433-7765
DSN 713-4418, COMM 937-255-4418
Email: Jeffrey.Hebert.1@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RYWN

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Monocular visual odometry (VO) has seen significant advances in the last decade, recently producing several real-time solutions that can be used for navigation without reliance on GPS. However, the majority of current research has focused on developing and applying these algorithms on ground-based or quadrotor platforms in constrained environments. This research compares the performance of three open-source and current state-of-the-art VO algorithms on a fixed-wing SUAS through simulation and real-world flight tests in large, outdoor environments and under high-speed maneuvers. The algorithms tested are Direct Sparse Odometry, Semi-Direct VO and ORB-SLAM2 with loop closures disabled.

15. SUBJECT TERMS

Visual Odometry, SUAS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. R. C. Leishman, AFIT/ENG
U	U	U	UU	93	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636; Robert.Leishman@afit.edu