Cooperative Navigation of Fixed-Wing Micro Air Vehicles in GPS-Denied Environments

Gary James Ellingson
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Engineering Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/8706

This Dissertation is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
Cooperative Navigation of Fixed-Wing Micro Air Vehicles
in GPS-Denied Environments

Gary James Ellingson

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Timothy W. McLain, Chair
Randal W. Beard
Cameron K. Peterson
Marc D. Killpack
John L. Salmon

Department of Mechanical Engineering
Brigham Young University

Copyright © 2019 Gary James Ellingson
All Rights Reserved
ABSTRACT

Cooperative Navigation of Fixed-Wing Micro Air Vehicles in GPS-Denied Environments

Gary James Ellingson
Department of Mechanical Engineering, BYU
Doctor of Philosophy

Micro air vehicles have recently gained popularity due to their potential as autonomous systems. Their future impact, however, will depend in part on how well they can navigate in GPS-denied and GPS-degraded environments. In response to this need, this dissertation investigates a potential solution for GPS-denied operations called relative navigation. The method utilizes keyframe-to-keyframe odometry estimates and their covariances in a global back end that represents the global state as a pose graph. The back end is able to effectively represent nonlinear uncertainties and incorporate opportunistic global constraints.

The GPS-denied research community has, for the most part, neglected to consider fixed-wing aircraft. This dissertation enables fixed-wing aircraft to utilize relative navigation by accounting for their sensing requirements. The development of an odometry-like, front-end, EKF-based estimator that utilizes only a monocular camera and an inertial measurement unit is presented. The filter uses the measurement model of the multi-state-constraint Kalman filter and regularly performs relative resets in coordination with keyframe declarations. In addition to the front-end development, a method is provided to account for front-end velocity bias in the back-end optimization.

Finally a method is presented for enabling multiple vehicles to improve navigational accuracy by cooperatively sharing information. Modifications to the relative navigation architecture are presented that enable decentralized, cooperative operations amidst temporary communication dropouts. The proposed framework also includes the ability to incorporate inter-vehicle measurements and utilizes a new concept called the coordinated reset, which is necessary for optimizing the cooperative odometry and improving localization.

Each contribution is demonstrated through simulation and/or hardware flight testing. Simulation and Monte-Carlo testing is used to show the expected quality of the results. Hardware flight-test results show the front-end estimator performance, several back-end optimization examples, and cooperative GPS-denied operations.

Keywords: GPS degradation, GPS denied, navigation, state estimation, observability, extended Kalman filter, sensor fusion, vision-aided INS, consistency, multirotor, fixed wing, micro air vehicle, indoor flight, outdoor flight, simultaneous localization and mapping, place recognition, loop closure, pose graph optimization, visual odometry, cooperative estimation
ACKNOWLEDGMENTS

Working in the MAGICC lab over the last several years has been a highly rewarding journey. I have learned much and grown significantly. I am first and foremost grateful for my Savior, Jesus Christ. He is the enabler and I only hope that I can be worthy to be called His disciple. I give thanks to my Father in Heaven for the gift of His Son and for His frequent guidance through the Spirit. I would be amiss if I did not thank my incredible wife, Katie Ellingson, for her patience, love, and constant support. She has maintained confidence in me even when I lacked confidence in myself. My parents and siblings have also been very supportive. I thank my advisor, Tim McLain, for continual support, advice, and mentorship and Kevin Brink for both his valuable guidance and interest in the success of this research. The many discussions, collaborations, and friendship with fellow researchers in the MAGICC lab, especially Dan Koch and James Jackson, have been invaluable to my happiness and intellectual development. It has been a blessing for me to interact with them and many others at this amazing university. Finally, I am grateful for the generous funding from the NSF-sponsored Center for Unmanned Aircraft Systems, AFRL Munitions Directorate, the BYU Mechanical Engineering Department, and the Utah NASA Space Grant Consortium.
# TABLE OF CONTENTS

List of Tables ........................................................................................................ vii

List of Figures .......................................................................................................... viii

Chapter 1 Introduction ............................................................................................. 1
  1.1 Micro Air Vehicles ............................................................................................ 2
  1.2 GPS-denied Navigation .................................................................................... 4
  1.3 Visual-Inertial Estimation ................................................................................ 5
  1.4 Cooperative Navigation .................................................................................. 6
  1.5 Problem Statement ......................................................................................... 6

Chapter 2 Contributions ......................................................................................... 7
  2.1 Relative Navigation Benefits and Experimentation ......................................... 7
  2.2 Visual-Inertial State Estimation For Fixed-Wing MAVs .............................. 8
  2.3 Robust Multi-Vehicle Cooperative Localization ........................................... 8
  2.4 Contributions ................................................................................................ 9
  2.5 Additional Works ......................................................................................... 11

Chapter 3 Hardware and Software Experimental Setup ..................................... 13
  3.1 Software ........................................................................................................ 13
    3.1.1 ROS and Gazebo .................................................................................... 13
    3.1.2 ROSflight Tools ................................................................................... 14
    3.1.3 Optimization ...................................................................................... 15
  3.2 Hardware ....................................................................................................... 15
    3.2.1 Multi Rotor ......................................................................................... 16
    3.2.2 Fixed Wing ......................................................................................... 17

Chapter 4 Relative Navigation Experimentation ............................................... 19
  4.1 Architecture .................................................................................................. 19
Chapter 6  Cooperative Relative Navigation of Multiple Aircraft in GPS-Denied/Degraded Environments ............................... 63

6.1 Introduction .............................................. 63
6.2 Related Works ............................................ 65
6.3 Development ............................................. 69
  6.3.1 Architecture ......................................... 69
  6.3.2 Inter-Vehicle Measurements and Messaging ............. 70
  6.3.3 Communication Protocol ............................... 71
6.4 Simulation ............................................... 73
  6.4.1 Description .......................................... 73
  6.4.2 Results ............................................. 75
6.5 Hardware Experiments .................................... 78
  6.5.1 Description .......................................... 78
  6.5.2 Results ............................................. 81
6.6 Post-Processed Experiments ............................... 83
6.7 Conclusion ............................................... 84

Chapter 7  Conclusion and Future Work ........................... 87

7.1 Assumptions and Limitations .............................. 87
7.2 Planning and Control ..................................... 88
7.3 Front-End Estimator ..................................... 89
7.4 Map Generation .......................................... 90
7.5 Back-End Measurements .................................. 91

References ................................................... 91
LIST OF TABLES

3.1 Multi-rotor hardware details. ............................................. 16
3.2 Fixed-wing hardware details. ............................................. 18

6.1 Edge message definition. ................................................... 72
6.2 Range measurement message definition. ............................... 72
6.3 Request message definition. .............................................. 73
6.4 Response message definition. ............................................. 74
6.5 Amount of communicated data during test. ............................. 84
LIST OF FIGURES

3.1 Gazebo simulation tools make developing autopilots more seamless. .................. 14
3.2 The multi-rotor vehicle used in the initial flight tests. ................................. 16
3.3 The fixed-wing vehicle used for the later flight tests. ................................. 17
4.1 Block diagram of the general relative navigation system architecture. .......... 20
4.2 Pose-graph map for the outdoor flight test. ............................................. 22
4.3 Pose-graph map for the indoor flight test. ............................................. 23
4.4 Pose-graph map for the indoor-outdoor transition flight test. .................... 24
5.1 The method was tested on a modified STRIX StratoSurfer, a 1.5 m wingspan aircraft. 27
5.2 The relative navigation architecture modified for fixed-wing UAS. ............... 28
5.3 An example of a relative-navigation graph that is able to accurately model uncertainty. 31
5.4 The coordinate-frame transformations necessary for the feature point optimization. 37
5.5 Simulated and real camera image with feature track histories. ...................... 40
5.6 The estimator sensitivity to unmodeled calibration, timing, and initialization errors. 42
5.7 Three simulated flight trajectories, showing the effect of straight-and-level flight on estimator performance. ......................................................... 45
5.8 True and estimated trajectory of the aircraft over a manually flown flight test. . 46
5.9 The true and estimate states of a UAS in flight relative to the most recent node. 46
5.10 State error and 3σ uncertainty bounds are shown for position, velocity, and attitude states. ................................................................. 47
5.11 Factor graphs, including bias factors, used in the global back-end. .............. 49
5.12 Back-end optimization of the flight-test graph with three simulated GPS measurement constraints. ......................................................... 51
5.13 Back-end optimization of the flight-test graph includes five simulated range measurements to two static features or DME stations. .................. 52
5.14 Back-end optimization of a graph including edges from three vehicles. ........ 53
6.1 The relative navigation architecture modified for multiple-vehicle operations. 68
6.2 Timeline and multi-vehicle graph motivating the concept of the relative reset. 69
6.3 Simulation results of three cooperative agents. 75
6.4 Monte-Carlo simulation results showing expected localization improvement provided by coordination. 77
6.5 Monte-Carlo simulation results showing the relative errors between the coordinating agents are reduced. 78
6.6 The method was tested on a 1.5 m wingspan aircraft that was coordinating with aircraft simulated from recorded data. 79
6.7 The resulting graph computed on board the test aircraft at the end of the 6-km flight. 81
6.8 The graph before and after communication was regained following a dropout. 82
6.9 Backend graph showing the benefit of additional measurement constraints. 83
CHAPTER 1. INTRODUCTION

“Ralphiebot, make me a sandwich!” It’s from a memorable scene in the kids’ TV show *The Magic School Bus*. An innovative student Ralphie has just built a robot and now is commanding it to do his bidding. The robot then promptly fails to follow Ralphie’s command. The robot is physically capable of constructing a delicious sandwich, but Ralphie is left hungry.

The vision of building an autonomous system is a compelling one. A system that can understand its environment and how to interact with it could be used for much more than making sandwiches. As they gain more and more capabilities, autonomous systems free the human users to focus elsewhere and perform other tasks, effectively increasing their efficiency. These systems require both physical capability to perform their task and, perhaps more importantly, intelligence to safely accomplish their objectives.

It is appropriate to consider autonomy as levels in a hierarchy. It is nearly impossible for there to be an agent that operates in the world without some interaction with other agents. In other words, every agent requires some level of supervision or interaction with human users or other agents for it to accomplish its objective. For a system to be considered to have higher levels of autonomy the system must requires less frequent and less precise supervision and must be more capable of making small, intermediate decisions about the best course of action.

Autonomous systems, in many forms, are on the verge of being let out into the wild. While robotic machines have been used for decades in controlled environments, such as factories and warehouses, recent increases in computational power and efficient algorithms are enabling robots to navigate in more chaotic environments. The promise of self-driving cars are a good example.
A robotic vehicle must be able to perceive the environment and model uncertainty about many potential unexpected situations to safely operate on public roads.

1.1 Micro Air Vehicles

Micro air vehicles (MAVs) are another example of an autonomous system whose time has nearly come. Although the basic concepts have been around for some time, MAVs are becoming increasingly ubiquitous in recent times. There has been an explosion in their popularity for public and hobbyist use because they are inexpensive and able to provide entertainment and a unique perspective of the world. MAVs have also been proposed for a variety of commercial applications including autonomous delivery, infrastructure inspection, and mapping. In military theaters, MAVs are being used for surveillance and prosecution of the enemy. Finally, MAVs have seen a recent increase in interest from both academia and commercial research and development. The focus has largely been on increasing their capabilities and raising their level of autonomy.

As with other aircraft, MAVs are simultaneously constrained by their size, weight, and power (SWaP) requirements. Especially on small aircraft, these constraints also limit the amount of computational resources that the vehicle can carry. Recent growth in MAV popularity has largely come from the optimization and miniaturization of transitional methods for achieving autonomy that now fit within the MAVs’ constraints.

MAVs have achieved the lowest levels of autonomy. They can effectively self stabilize, hover in place, and can follow a trajectory or a waypoint path. Thus far, regulatory constraints due to MAV reliability have largely limited their operation to visual line-of-sight, where the user can utilize the MAV as a tool but with near constant operation, direct control, or careful monitoring.

There are many names for a small, pilot-less aircraft. Throughout this dissertation they will be called micro air vehicles (MAVs), unmanned air vehicles (UAVs), and unmanned aircraft systems (UAS). All these names refer to a small vehicle that has some ability to fly under its own power. A
large variety of MAVs exist, including helicopters, rockets, and airplanes as well as some exotic examples: lighter-than-air vehicles (such as balloons and blimps) and mechanical ornithopters. Two main categories are considered in this work: fixed-wing (FW) MAVs and multi-rotor (MR) MAVs. While the categories are not completely rigid, these vehicle types have important differences and distinctions.

MRs are usually recognized by four or more spinning propellers. Often fixed-pitch propellers are used and the relative speed of each propeller is modulated to produce torques on the vehicle body. They have the distinct ability to hover in place. Indeed actuation of MR MAVs is often characterized by slight deviations from the hover condition, by either changing the total thrust to climb or descend or by inducing a small torque on the body to make it tilt in the desired direction of travel. Most recent research and development has been on MRs, especially when considering GPS-denied scenarios.

FWs are characterized by a relatively large wing surface and a propeller, fan, or jet that produces forward thrust. To remain airborne, they rely on air moving over the wing to produce lift. A neutral condition is straight-and-level flight with constant airspeed. Actuation is enabled through changing the propeller speed, and thus thrust, or by deflecting control surfaces that change aerodynamic forces and induce torques on the body of the aircraft. FW MAVs have significant lateral-directional dynamics that are not found in MR. This means a FW cannot travel in an arbitrary direction without first rolling, coordinating a turn, and ultimately pointing in the general direction of travel.

The flight profile of FWs are usually very different from MRs. MRs can easily fly in an around obstacles in the environment because they have the ability to hover in place and fly slowly. FWs however are more suited to fly fast and far, where being high above the environment is advantageous.
1.2 GPS-denied Navigation

One reason that the autonomous capabilities of MAVs have been limited is because traditional autopilot technology relies heavily on GPS measurements. While navigation solutions exist that only utilize inertial measurement units (IMUs), these solutions require extremely accurate and expensive, tactical-grade IMUs that are generally not feasible for MAVs. It is much more common for MAV to fuse GPS with inertial measurements (GPS-INS) to mitigate drift due to sensor errors. Traditional GPS-INS methods benefit from combining high rate and high noise IMU measurements with low frequency and low noise GPS measurements.

Optimization and miniaturization of MAVs have enabled them to operate in a variety of new and advanced ways that make relying on GPS impractical. Small MAVs are capable of navigating in confined spaces, urban environments, and inside buildings where GPS signals are unavailable. There are many civil applications, such as autonomous drone delivery services and infrastructure inspection, that require MAVs to operate during GPS degradation and dropout due to proximity to buildings and obstacles. In military theaters, where MAVs are being used to observe and prosecute the enemy, GPS is often spoofed or jammed and MAVs need to perform the same functions without relying on GPS measurements.

One of the main difficulties of GPS-denied navigation is being able to accurately model navigational uncertainty while keeping the method computationally tractable. Without geo-registered updates to remove the drift that accumulates over time, global position and yaw angle are unobservable [1, 2, 3]. Often, traditional navigation methods use a linear Gaussian approximation of the uncertainty to make the problem computationally tractable [4]. These methods become inconsistent as errors grow and can struggle to accurately remove errors when measurements become available.
1.3 Visual-Inertial Estimation

In an attempt to make up for the lack of GPS, additional exteroceptive sensors can be added to the vehicle and their measurements fused in a filter. Cameras, depth sensors, and laser scanners can be used on MAVs to measure the motion of the vehicle relative to the environment and make the estimated motion more accurate.

Depth sensors can be advantageous for navigation of MAVs and have been used extensively in various GPS-denied solutions. They are impractical, however, for FW MAVs because FW generally fly high above the environment, beyond the effective range of the sensor, or because the sensors with adequate range are too large and heavy for the vehicle to carry. Further, if a MAV is high above the approximately flat environment, the depth measurements alone are not well-suited for measuring the motion of the vehicle.

Cameras, which are easily carried by MAVs, can be used as sensors for navigation using visual odometry [5, 6]. Position and velocity measurements can be obtained by tracking features across image frames and using the feature tracks to solve for the camera motion. Odometry can be calculated frame-to-frame or with respect to a keyframe. Keyframe-based approaches have the advantage of reducing temporal drift in the odometry measurements [7].

Visual navigation approaches can be categorized by how tightly the visual and inertial measurements are integrated or coupled. Some approaches do not incorporate inertial sensors [8], but suffer from scale ambiguity. In decoupled approaches visual and inertial navigation solutions are computed separately and simply compared [9, 10]. Loosely coupled approaches compute the visual solution separately, but combine the measurements in a filter. Tightly coupled approaches, known as visual-inertial odometry (VIO), use the current navigation solution to inform the visual estimates and image feature tracks that directly feed into the navigation filter.
1.4 Cooperative Navigation

MAVs can benefit from cooperatively sharing information. There are several examples that are found in the MAV literature [11, 12, 13]. They can benefit from cooperation by coordinating to achieve a mission objective by subdividing the task. Obstacle avoidance and conflict mitigation can be enabled by MAV broadcasting their navigational states and intentions for other aircraft to deconflict and avoid their path.

In GPS-denied navigation, MAVs can improve their navigational accuracy by sharing information, such as sensor measurements, to improve their estimates. There are several challenges and constraints, however, to enabling coordination of MAVs. First the MAVs must have a method of communication that is robust. Then, they must have a method for fusing their information to improve their estimates in a computationally efficient way. Finally, the MAVs need to be able to incorporate the improved estimates to improve their ability to achieve their objective. A variety of methods exist for cooperation, each with benefits and drawbacks.

1.5 Problem Statement

Like Ralphibot and other autonomous systems, MAVs need the ability to both perceive the environment and accurately model their uncertainty and objective. Doing this will increase their level of autonomy and allow them to be leveraged efficiently by human users. Recent advances in MAV technology have increased the need for robust GPS-denied navigation capabilities. The objective of this work is to develop a GPS-denied navigation approach for small, inexpensive, fixed-wing aircraft while accounting for the vehicle’s flight profile and sensing requirements. The approach will be applied to several MAVs that are cooperatively navigating and have the ability to incorporate opportunistic global measurements. The goal is to show that several inexpensive MAVs can cooperatively navigate as well as a high-cost, high-accuracy solution. Finally, this dissertation includes an extensive focus on the experimental validation of the proposed methods.
CHAPTER 2. CONTRIBUTIONS

This chapter summarizes the contributions made within this dissertation. Much of the research has been published, is currently under review to be published, or is being prepared for future publication. In two instances, journal papers are included as chapters in this dissertation.

2.1 Relative Navigation Benefits and Experimentation

This dissertation begins by introducing relative navigation and discussing several significant hardware flight tests that were performed as part of the development and validation of the relative navigation framework. The framework has several advantages for GPS-denied navigation. It was originally proposed as a modular framework that would be flexible for multiple vehicles in differing environments.

Two papers were published in coordination with several other authors. The flight test results from the papers are discussed to give the initial context for the relative navigation architecture. My roll in the research mainly consisted of enabling the flight testing by creating an embedded autopilot for low-level stabilization, providing access to the various sensors, and performing the actual test flights. The papers are


2.2 Visual-Inertial State Estimation For Fixed-Wing MAVs

Next, the framework is extended to fixed-wing aircraft. Although the principles of the relative navigation framework are valid for the fixed-wing aircraft, their estimation and sensing requirements are different from the multi-rotor MAVs used in the previous experiments. These differences motivate the development of a tightly-coupled visual-inertial state estimator that does not rely on depth measurements and makes no assumption about the distance to observed image features. Both simulation and hardware flight test results are provided to show the estimator is capable of enabling the relative navigation architecture on fixed-wing aircraft. The development details, and a discussion of several sources of error that must be accounted for to obtain the flight test results, are presented in the following paper that makes up Chapter 5 of this dissertation:


2.3 Robust Multi-Vehicle Cooperative Localization

The relative navigation architecture has several properties that are valuable for cooperative multi-vehicle operations. First, the front-end estimator is able to compress the high-rate sensor information into relatively low-frequency odometry transforms. This property enables sharing information between MAVs feasible even with low bandwidth or intermittent communications. Next, the back-end graph optimization is able to incorporate information that is delayed, which also helps with intermittent communications. The relative-navigation architecture, however, must be modified to enable inter-vehicle communication and measurements between cooperating MAVs. The necessary modifications, as well as simulation and hardware demonstrations of multiple aircraft, are presented in Chapter 6, which has also been submitted for publication as
2.4 Contributions

To precisely navigate MAVs without relying on global measurements, such as GPS, it is vital to accurately represent the highly nonlinear navigational uncertainty in a computationally efficient way. Additionally, GPS-denied navigation of fixed-wing MAVs is uniquely challenging because they are well-suited to fly long, straight trajectories high above the environment where many exteroceptive sensors are unable to provide accurate measurements of the aircraft motion. Furthermore, cooperative navigation is difficult, especially when inter-vehicle communication is limited or unreliable. This dissertation provides several key technical contributions for enabling GPS-denied navigation of MAVs.

- It presents results of significant flight tests that introduce, highlight, and validate the practical and theoretical advantages of the proposed relative navigation framework for operating MAVs in a variety of GPS-denied/degraded environments. The approach incorporates the computational efficiency of an extended Kalman filter that is built to operate in real time and to be observable by construction. Further, the framework includes a back-end that can accurately represent the nonlinear uncertainty, robustly utilize opportunistic constraints, and accommodate slower-than-real-time graph optimization when necessary.

- It describes a method that enables GPS-denied operations of fixed-wing MAVs with a new tightly-coupled visual-inertial odometry method that draws on the relative navigation principles. This was not previously possible due to the reliance on depth sensors, which are impractical on fixed-wing MAVs due to their high-altitude flight profiles. The proposed odometry utilizes the measurement model of the multi-state-constraint Kalman filter (MSCKF), which was chosen because it fits within in the relative navigation paradigm, has accuracy performance that is comparable to the state of the art, is computationally efficient, and allows
for fixed-wing specific flight profiles and sensing requirements. The stochastic properties of the MSCKF are also improved by incorporating the reset step from the relative navigation framework. The method thus produces consistent and observable state estimates for fixed-wing aircraft in the absence of GPS.

- This research introduces two important enhancements to the relative navigation architecture. First, it details a method that accounts for the correlation between edges by introducing a bias factor in the back-end graph. Previously graph edges were modeled as independent, despite the velocity state persisting through the relative reset step. The errors in the velocity states accumulate to produce a slowly varying scale bias in the pose graph. Accurately modeling the scale biases in the back end enables the optimization to remove their effects when additional constraints are available. The proposed method for modeling bias is valuable for relative navigation generally but is essential for relative navigation of fixed-wing MAVs because their nominal flight condition is straight-and-level flight where the forward velocity is unobservable with visual-inertial odometry alone. Second, it incorporates a method for coordinating between MAVs. Relative navigation has several characteristics that are beneficial for cooperative GPS-denied operations, including its scalability and opportunistic nature. The communications architecture developed in this research incorporates odometry from multiple agents into a decentralized optimization. Further, the communications architecture enables inter-vehicle measurements that constrain the optimization by coordinating front-end resets with measurements.

- This dissertation documents significant flight demonstration results that illustrate real-time single-vehicle GPS-denied operations and multi-vehicle cooperative GPS-denied navigation. The accuracy of the front-end odometry is shown to be comparable to the state-of-the-art visual-inertial odometry. The multi-vehicle results demonstrate a method that is robust to communication dropout and delay and that provides a significant improvement to both the global navigation accuracy and the relative accuracy of the formation.
2.5 Additional Works

This dissertation does not discuss several additional publications that I have authored as a graduate student. These works represent significant projects that were performed as part of my graduate studies. While they are generally outside the body of this work and do not contribute to the specific topic of this dissertation, they have been an important part of my graduate experience and I have chosen to briefly summarized them here.

Two conference papers were published that use deep learning for MAV attitude estimation. The main concept for these papers was to use a deep-convolutional neural network (CNN) as sensor in a more classical estimation framework.


The first paper utilized an on-board camera, a CNN, visual odometry, and an EKF to produce aircraft attitude estimates. The second used a remote camera and performed third-person-perspective estimation with a mixture-of-Gaussian tracker, a CNN, a particle filter, and a relatively simple two-dimensional propagation model. Both papers provide examples of how the CNN is bio-inspired. That is, the CNN is able to accurately mimic human capabilities to measure something that would be difficult to algorithmically quantify with classical methods alone.

Two additional works were published to present the details of a hardware and software framework for rapid experimentation and prototyping of autopilot technology. Some of the provided tools, specifically the simulation framework, were used in the development of this dissertation.
The ideas have also been improved and extended and have been used in organizations throughout the world.


**ROSplane: Fixed-wing autopilot for Education and Research.** *Ellingson and McLain.* Published at the International Conference on Unmanned Aircraft Systems (ICUAS) in June 2017.

The first paper uses hobbyist flight-control hardware, commonly used on first-person-view quadcopters, to perform low-level estimation and sensor input-output for an embedded companion computer running the robot operating system (ROS). The second paper utilizes that capability to create a fully functional fixed-wing autopilot that is based on [14]. It was created to be used by students taking a course on MAVs and has been used, together with ROSflight, by students in the AUVSI student competition.

Finally, a paper was published that introduced a method for autonomous landing-zone determination. The premise of the project included a multi-rotor MAV used for surveillance of a target. To increase the mission beyond the maximum flight duration, the MAV would autonomously locate a suitable rooftop on which it could perch while observing the target.

CHAPTER 3. HARDWARE AND SOFTWARE EXPERIMENTAL SETUP

This chapter outlines the main software tools and hardware components utilized in the various experiments presented throughout the remainder of this work.

3.1 Software

3.1.1 ROS and Gazebo

The Robot Operating System (ROS) [15] is a software framework and middleware commonly used in many robotic applications and includes many standardized robotic development tools. Included are tools for processing a variety of sensor information, simulating robotic hardware, and developing robotic software. ROS utilizes a publisher/subscriber framework where every task is represented by and implemented as a node in a directed graph structure with inputs and outputs to other nodes. ROS has also been applied to research of autonomous aircraft [16, 17].

ROS provides important features used in the development and experimental testing of this research. Because it provides standardized interfaces through message definitions, it can interface between various programming languages, namely C++ and Python. Both languages are used extensively throughout this work. ROS also provides tools for recording and replaying messages. These tools were vital to the experiments presented in Chapter 6. Finally, robotic sensors are often distributed with a driver node that directly interfaces with the sensor hardware and provides the measurements in standardized message formats.

Gazebo is a robotic simulation framework that was developed separately but alongside ROS, and is capable of tight integration with ROS. It includes standard robots for complete physical
Figure 3.1: Simulation tools utilizing Gazebo plugins make developing autopilot more seamless by allowing the hardware driver to be switched with a simulation node in ROS. A hardware configuration (left) is nearly identical to the simulation configuration (right). Blocks that are bold with rounded corners are ROS nodes and blocks with square corners represent physical computing devices.

simulation, as well as standard environments for them to interact with, including items from an ordinary traffic cone to the international space station. Gazebo also provides flexibility to incorporate custom software plugins for defining new sensors and robots. The Gazebo simulation tools have been used throughout the development and testing of this research.

3.1.2 ROSflight Tools

ROSflight [18] is a input/output (I/O) board, firmware, and driver for ROS-based autopilots. It was created to facilitate research-autopilot development. It utilizes open-source hardware and software to create an inexpensive, readily-available flight-control-unit I/O board. The I/O board abstracts the real-time critical processes such as sensor and receiver reading and actuator output. It allows autopilot estimation and control to happen directly in ROS, thus avoiding complex embedded development where possible.
The ROSflight system also includes aircraft simulation tools that interface ROS with a Gazebo simulator. The embedded ROSflight firmware can be compiled into a Gazebo plugin to create a software-in-the loop simulation. Figure 3.1 show how Gazebo can be exchanged for a physical vehicle while leaving the rest of the architecture the same. The ROSflight tools together with a custom ROSplane [19] Gazebo simulation were also used for development and testing.

3.1.3 Optimization

This work also used the Levenberg-Marquardt (LM) nonlinear least squares algorithm implemented in a software library called levmar [20] for feature point optimization. LM is a combination of steepest descent and the Gauss-Newton methods and provides a useful combination of convergence guarantee and speed of execution. Levmar utilizes optimized linear algebra libraries LAPACK/BLAS for performing the optimization. It can perform both single and double precision optimization with analytic or finite-difference approximated Jacobians. It also can incorporate linear or box constraints and has both a C and C++ interface.

This work also utilizes pose transformations as factors in factor graph smoothing frameworks, such as generalized graph optimization (g2o) [21], Georgia Tech smoothing and mapping (GTSAM) [22], and incremental smoothing and mapping (ISAM) [23]. These frameworks have open-source implementations that are provided for research purposes. The back-end optimization results provided in Chapter 4 use g2o, while the rest of the back-end optimizations are performed using GTSAM.

3.2 Hardware

There are two main platforms used in the hardware flight testing. Chapter 4 utilized a multi-rotor MAV and Chapters 5 and 6 utilized a fixed-wing MAV. The major components of these platforms are outlined in the following sections.
Figure 3.2: The multi-rotor vehicle used in the initial flight tests.

Table 3.1: Multi-rotor hardware details.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>Hexacopter, 4.8 kg, 0.69 m diameter</td>
</tr>
<tr>
<td>Autopilot</td>
<td>3DR Pixhawk</td>
</tr>
<tr>
<td>RGB-D Camera</td>
<td>ASUS Xtion Pro Live</td>
</tr>
<tr>
<td>Laser Scanner</td>
<td>Hokuyo UTM-30LX</td>
</tr>
<tr>
<td>IMU</td>
<td>MicroStrain 3DM-GX3-15</td>
</tr>
<tr>
<td>Altimeter</td>
<td>I2CXL-MaxSonar-EZ MB1242</td>
</tr>
<tr>
<td>GPS</td>
<td>U-blox LEA-6T</td>
</tr>
<tr>
<td>Processor</td>
<td>Intel Core i7-2710QE (2.1 GHz ×4)</td>
</tr>
</tbody>
</table>

3.2.1 Multi Rotor

The multi-rotor platform used in initial testing is a highly modified HexaKopter by Mikrokopter, shown in Figure 3.2. The vehicle weights 4.8 kg when loaded and uses six MK3638 brushless motors and 12×4.5 propellers. It is powered by two parallel 5000 mAh four-cell lithium-polymer batteries.

The vehicle carries an Intel Core i7 computer running ROS and interfaces with a Pixhawk autopilot and a variety of sensors. The Pixhawk runs a custom low-level estimator and controller for attitude stabilization and is responsible for providing the motor commands based on inputs from the ROS computer. Challenges with the use of the Pixhawk motivated the development of ROSflight [18] mentioned earlier. Several depth sensors are included on the vehicle. It holds a RGB-D camera that provides a dense point cloud along with a monocular image and a single scan.
Figure 3.3: The fixed-wing vehicle used for the later flight tests. In the lower left image the computer, IMU, and camera have been removed and set beside the vehicle.

The fixed-wing flight tests utilized a STRIX StratoSurfer, a 1.5 m wingspan aircraft by Ready Made RC, shown in Figure 3.3. The aircraft is driven by a B2212-2200 kV motor, $6 \times 4$ propeller, and 2200 mAh three-cell lithium-polymer battery.

The vehicle contains a NVIDIA Jetson TX2 embedded computer that includes an integrated GPU for parallelizing image processing. The aircraft also utilized an InertialSense micro inertial navigation system ($\mu$INS) for both raw IMU and GPS-INS truth comparison. Finally, a camera is carried on board the aircraft and points 45 degrees downward from the center axis of the vehicle. The computer runs ROS for interfacing with sensors and GTSAM for graph optimization. The camera and IMU measurements both receive accurate time stamps from the $\mu$INS. This is done by configuring the camera to emit a strobe pulse that coincides with the camera shutter. The pulse is
Table 3.2: Fixed-wing hardware details.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>StratoSurfer, 1.4 kg, 1.5 m wingspan</td>
</tr>
<tr>
<td>Camera</td>
<td>Point Grey Chameleon 3, 1.3 MP</td>
</tr>
<tr>
<td>IMU</td>
<td>InertialSense Development kit</td>
</tr>
<tr>
<td>GPS</td>
<td>InertialSense μINS</td>
</tr>
<tr>
<td>Radio</td>
<td>3DR 916 MHz SiK telemetry</td>
</tr>
<tr>
<td>Processor</td>
<td>ARM A57 (1.2 GHz ×4) &amp; Denver 2 (1.2 GHz ×2)</td>
</tr>
<tr>
<td></td>
<td>NVIDIA Pascal GPU (×256)</td>
</tr>
</tbody>
</table>

used by the μINS to produce an accurate time stamp for the image and the image measurements are combined with their time stamps once they are both processed in ROS. The hardware components used on the vehicle are summarized in Table 3.2.
CHAPTER 4. RELATIVE NAVIGATION EXPERIMENTATION

This chapter introduces the relative navigation architecture and its advantages for GPS-denied navigation. It then provides several examples of flight-test experiments that demonstrate the architecture. The flight tests were performed as early validating examples of several aspects of the architecture. The tests were performed in collaboration with several researchers and the results were published in [24].

4.1 Architecture

In recent years, researchers in the BYU MAGICC Lab have been working on an approach to the GPS-denied problem. The approach is called relative navigation and initial concepts were published in [25] and [26]. It is a methodology and framework for GPS-denied navigation. It includes a front end that estimates the relative state and navigates relative to the local surroundings, and a back end that optimizes the relative information into global state estimates. Figure 4.1 shows the general relative navigation architecture.

The front-end includes an extended Kalman filter (EKF) for state estimation. The EKF used in the early tests is fully defined in [27]. One of the main concepts of the filter is that it estimates its position relative to the local environment. It does not contain a global estimate and it regularly resets its position and heading to zero. This requires that the front-end controller also control the vehicle path in the local coordinate frame.

For results in this section the estimator utilizes a keyframe-based odometry as a measurement of the vehicle motion. Specifically, the depth-enhanced monocular odometry algorithm in [28] and the laser-scan matcher algorithm in [29] were used. The estimator combines the motion mea-
Figure 4.1: Block diagram of the general relative navigation system architecture. Real-time, flight-critical estimation and control is performed relative to a local frame. Global mission planning and localization are performed in the back end.

surement with separate IMU measurements. This means the front-end estimation scheme is a loosely-coupled visual-inertial odometry.

The architecture includes a back end that is responsible for accounting for the global information. It takes regularly published odometry from the front-end estimator and represents the trajectory as a pose graph where poses are nodes connected by odometry edges. The graph can then be optimized as it receives additional constraints. Additional constraints can take many forms but usually contain opportunistic global information that helps remove accumulated drift form the trajectory.

4.2 Advantages

The work in [30] shows that the relative navigation architecture has several advantages over other commonly used GPS-denied estimation techniques. The advantages come from the fact that without regular GPS measurements both the global position and heading are unobservable. Many filters, including EKFs, make a Gaussian approximation of the position uncertainty, but as error grows the Gaussian is not able to accurately represent the uncertainty as heading errors make the true uncertainty look like a banana distribution. Relative navigation subverts this problem by
requiring the front-end estimator to reset its position periodically and representing the trajectory with a pose graph. This effectively makes the approach observable by construction (assuming the odometry position is locally observable).

If opportunistic global measurements are available, but intermittent, then incorporating them into a global EKF can be problematic. If a large amount of error has accumulated, then there will be a large update to the estimated position of the vehicle. Because the vehicle is likely using these states to calculate a control effort, there will also be a sudden jump in the control that can cause instability or undesirable response. The relative navigation architecture does not use global states to calculate control effort in the front-end and, since the front-end does not get updated by global information, it also provides the benefit of avoiding large control jumps.

4.3 Flight Test Experimentation

Three environments were used in the flight testing. Each flight test was used to highlight and explore different aspects of the relative navigation architecture. The experiments were conducted outdoors, indoors, and with indoor-outdoor transitions. The flight tests and their results are discussed in the following sections. Each flight test was flown on the multi-rotor platform discussed previously.

4.3.1 Outdoors

Because of the limitations of the RGB-D sensor used for visual odometry, the outdoor flight test took place at dusk and close to the ground and to buildings. The proximity to the building made GPS signals unreliable for the duration of the flight due to satellite occlusion and multi-path effects. The flight trajectory was 320 m in length and took approximately 9 minutes. Figure 4.2 shows the odometry graph compared to the true trajectory.
Figure 4.2: Pose-graph map for the outdoor flight test with a multi-rotor MAV. Heading errors cause the position uncertainty to grow. The global back end compounds the small, Gaussian edge covariances to form banana-shaped uncertainty estimates that correctly represent the underlying uncertainty. The 90 percent confidence regions are shown for several instances throughout the trajectory.

The flight test was able to show the capability of the relative-navigation architecture to represent uncertainty in the trajectory. The back-end graph can be used to approximate global pose uncertainty by taking the Gaussian covariance from each graph edge and compounding them using a 4th-order approximation of the banana distribution in exponential coordinates [31]. The 90 percent confidence regions are plotted in Figure 4.2 to show the graph is consistent when a simple ellipse would not be. Additional details about the banana distribution can be found in [30].

4.3.2 Indoors

This test utilized the multi-rotor MAV flying down several hallways in a large building. The several hallways together made a circuit, or complete path for returning to the original location. The flight trajectory was 390 m in length and lasted approximately 12 minutes. Because the flight was indoors, GPS was entirely unavailable and as expected the trajectory began to drift from the true path as the length of the flight increased. Figure 4.3a shows the raw flight odometry in the graph structure overlaid on the building floor plan.
One of the objectives of this test was to incorporate loop-closure constraints into the back-end graph as the MAV returned to previously visited locations. An open-source place-recognition software called fast, appearance-based mapping (FAB-MAP) \[32\] was incorporated. The results in Figure 4.3c show that the pose graph is significantly improved by optimizing and taking loop-closure constraints into account.

Although none were observed in the flight test, place-recognition methods are prone to false positives. For this reason a method was introduced for rejecting outliers when they were artificially introduced to the graph. Figure 4.3b shows the negative effect they have on the result when they are not rejected and Figure 4.3c shows the ability to reject them and produce the consistent trajectory.

### 4.3.3 Indoor-Outdoor Transitions

A final test was performed to show the architecture’s ability to transition from indoor-to-outdoor environments. This test differed slightly from the other tests because it utilized the laser-scan matching for odometry instead of the visual-odometry. The test consisted of a 240 m trajectory that took 9 minutes to fly. In this test the multi-rotor MAV started in a garage and exited the
building through a large roll-up door. The MAV makes several re-entries and exits throughout the remainder of the flight.

Global constraints in the back-end were able to improve the estimated flight trajectory. Place recognition was able to provide 30 loop closure constraints that removed the drift that had accumulated. When the vehicle was outdoors it was able to also obtain ten opportunistic GPS measurements of sufficient accuracy. The measurements were also incorporated in the graph and allowed it to be globally localized. Figure 4.4a show the raw unoptimized trajectory and Figure 4.4b shows the final optimized trajectory. The GPS measurements were all biased north due to the urban canon nature of the buildings and thus the final graph is also biased.

Additionally the test demonstrated the ability of the vehicle to fly in a fully autonomous mode. A set-point was given to the aircraft and it was able to use a previously flown path to autonomously enter a building. As the aircraft entered the building it also found a loop closure and subsequently
optimized the map. The optimized position did not effect the front-end estimation or control and thus large control jumps were effectively avoided.

4.4 Summary

The research presented thus far has set the stage for the fixed-wing relative navigation work that is described in Chapter 5 and the multi-vehicle coordination work in Chapter 6. The major aspects of the framework have been demonstrated, including front-end estimation, back-end optimization, opportunistic global measurements, and fully autonomous flight. The relative navigation framework is proposed as a general approach that can be adapted to other vehicles, sensors, and environments [33]. Every experiment provided to this point, however, has utilized a multi-rotor MAV that has relied on depth sensors for measuring the aircraft motion. Further, the trajectories have been limited to relatively short, looping paths that were flown in close proximity to the ground as well as in and around structured environments. The following chapters describe important extensions to the work presented thus far that extend its impact to fixed-wing aircraft and teams of fixed-wing aircraft.
CHAPTER 5. RELATIVE NAVIGATION OF FIXED-WING AIRCRAFT IN GPS-DENIED ENVIRONMENTS

5.1 Introduction

The capabilities of unmanned aircraft systems (UAS) have dramatically increased over the past decade. This expansion of capabilities has largely been enabled by the development, optimization, and miniaturization of traditional navigation methods, where GPS measurements are fused with inertial measurements (GPS-INS). Figure 5.1 shows an example of a small, fixed-wing UAS. As UAS continue to get smaller and more advanced they will be able to operate in confined spaces, in urban environments, and inside buildings. For UAS to continue to expand to more applications they will require the ability to navigate when GPS is unavailable or unreliable. For example, many civil applications, including delivery and inspection services, require that UAS fly in close proximity to structures. Structures can reduce the accuracy and reliability of GPS signals. UAS can also be used in military applications for observing and prosecuting the enemy, but the threat of spoofing and jamming of GPS signals provides motivation for navigating without relying on GPS measurements.

Inertial measurements, by themselves, can be used to estimate the motion of a UAS, but sensor errors will accumulate and cause the estimates to drift. A UAS can be augmented with exteroceptive sensors, such as cameras or laser scanners, to measure the motion of the vehicle with respect to the surroundings. By fusing the inertial and exteroceptive measurements the motion estimate can be improved. In the absence of GPS or other global measurements, the global position and yaw angle are unobservable [1, 2, 3] and the estimates will eventually diverge.

1Submitted to Navigation: Journal of the Institute of Navigation
Figure 5.1: This work enables GPS-denied flight of fixed-wing UAS. The method was tested on a modified STRIX StratoSurfer, a 1.5 m wingspan aircraft.

Sensor noise filtering and measurement fusion can take place in an extended Kalman filter (EKF). EKFs, which are used extensively on robots [4] and UAS [34, 14] alike, account for both sensor errors and process uncertainty. They utilize a linear Gaussian representation of the state belief to take advantage of the computational convenience of a Kalman update but maintain the nonlinearity of the process propagation. This combination of properties performs well when errors remain small, such as when the availability of GPS measurements is used to regularly remove drift errors. The nonlinear nature of the process, however, causes the Gaussian representation of the belief to become inconsistent when errors are large due to the global states being unobservable and their estimates drifting from the true value. If a global measurement is received by an EKF after significant drift errors have accumulated, nonlinearities can make utilizing the measurement problematic. This causes over confidence, especially in states such as velocities, and IMU biases [30, 35]. This may result in large jumps in the estimate and, in severe cases, it can even cause filter divergence. Often, methods to allow EKFs to handle sparse opportunistic global measurements are ad hoc or cumbersome, including reinitializing the filter by shifting its origin, treating GPS as a relative sensor by transforming the measurement into a temporary coordinate frame [35] or gating (and thus ignoring) the measurement [36]. Some methods simply avoid using an EKF when GPS measurements are intermittent [37].
These observability and consistency problems have been addressed in recent years by the proposal of a new approach called relative navigation [26, 33]. Relative navigation has been introduced as a solution for operating UAS when GPS is either unavailable or intermittent at best. It utilizes an EKF for front-end estimation relative to the local environment and a back-end optimization that combines the relative information to produce the global estimates. The complete architecture is shown in Figure 5.2. Dividing the architecture into a relative front end and a global back end provides important observability and computational advantages. The front end navigates with respect to a local frame where the states can remain observable and the Gaussian distribution can accurately represent uncertainty, thus enabling the computational advantage of an EKF to be utilized. The back end uses a pose graph that can accurately represent nonlinearities in heading and be robustly optimized when given additional constraints. These constraints arise from measurements that constrain the pose either to a global location, such as an opportunistic GPS measurements, or, as introduced in this paper, relative to non-stationary objects, such as other aircraft.

The majority of GPS-denied navigation research has been performed with multirotor UAS as the experimental platform. To effectively navigate fixed-wing UAS without GPS, additional considerations including aircraft dynamics, mission profiles, and sensing requirements must be taken.
into account. Prior work that has focused on fixed-wing UAS has often either made significant simplifying assumptions, including flat-earth or Manhattan world environments, or imposed strict sensing requirements, such as a downward facing camera or distance measurements [38, 39]. The ability of multirotor UAS to hover in place, enables them to more-easily fly in and around buildings and structure. This allows laser scans and other distance measurements to effectively measure the aircraft motion. For fixed-wind UAS that typically fly at high speeds and high above the ground, the vehicles are often at altitudes that exceed or approach the limit of the measurement range of depth sensors making them less effective for sensing aircraft motion.

Using the relative navigation framework as a guide, this work enables GPS-denied navigation of fixed-wing UAS by developing a tightly-coupled, EKF-based, visual-inertial odometry (VIO). With the fixed-wing requirements in mind, we avoid the use of depth sensors, such as laser scanners and RGB-D cameras, and utilize only a monocular camera with no assumptions about the distance to observed features. By producing keyframe-based estimates of the change in pose, the front-end estimator enables the fixed-wing aircraft to utilize all the advantages of the global back end within the relative-navigation framework for GPS-denied navigation. This paper extends our previous efforts [40] where the concepts for the VIO filter and limited simulation results were initially presented. This paper provides a complete filter development and improved simulation results, as well as hardware, flight-demonstration results. Along with the flight-demonstration results, our efforts to mitigate calibration, timing, and initialization errors are discussed. Another contribution of this paper is in the back-end pose graph optimization. We provide a model of a slowly varying scale bias to account for both scale errors that arise from potentially unobservable velocity associated with straight-and-level flight and the correlation from one graph edge to the next. The full system localization is demonstrated by utilizing the front-end odometry together with various other opportunistic measurements that provide loop-closure-like constraints. Finally, to motivate future work, results are presented using measurements and constraints between cooperative aircraft that
demonstrate the potential of the proposed method for low-bandwidth, multi-vehicle cooperative localization.

5.2 Related Works

This paper builds upon previous research in two main areas: The overarching framework draws from the relative navigation body of research, and the method for constructing the visual-inertial odometry uses the principles from the multi-state-constraint Kalman filter (MSCKF). Relevant research contributions in these areas are summarized in the subsequent sections.

5.2.1 Relative Navigation

Relative navigation is built on an elegant concept: at any point in time an agent can have complete confidence in its position if, at that instant, it places its reference-frame origin at the vehicle center. An agent can further maintain good confidence in its local motion by observing the apparent motion of the local surroundings, even if the global position is unknown or is unobservable over large scales. As an example, a robot agent can set its initial position to zero and then localize around this initial origin, even though the origin’s global position is arbitrary.

Relative navigation uses this concept in the front-end filter in a process called the relative-reset step. The reset step is closely related to the keyframe update of keyframe-based odometry methods. As the vehicle travels from the current origin, the front-end filter is able to reset the origin to the current location of the vehicle (the new coordinate frame being aligned with the heading of the vehicle but level with the local-level frame), where the reset coincides with the declaration of a keyframe image. Within the EKF, the covariance associated with the position and heading states can then be zeroed and the states continue to evolve with respect to the newly-declared reference frame. The state from just prior to the reset then forms a transformation from one reset to the next and, together with the associated covariance, is provided to the back end. The transformations form a directed pose graph, where each origin is a node (or node frame) and each transformation
Figure 5.3: An example of a relative-navigation graph. Each edge (black) provides a transformation from one keyframe (or node) to the next. Although the transformations and associated covariances (purple ellipses) are linear, the graph is able to represent more complex, nonlinear uncertainties (such as the red banana distribution of the gray Monte-Carlo samples) better than a single Gaussian (blue) [3, 30]. This figure from [30] is reproduced with permission.

is an edge. Because the EKF operates only with respect to a local origin, it is observable, as well as consistent, by construction [30]. The uncertainty is regularly removed from the filter while a Gaussian is still able to accurately represent it, and non-linearities are handled appropriately in the back-end graph.

The global position and heading are accounted for in the back end because it contains the keyframe-to-keyframe transformations as edges in a pose graph. The global pose, which is necessary for accomplishing a mission with a global goal, can be produced by combining, or composing, the transforms. Figure 5.3 demonstrates how the graph edges are able to represent the nonlinear coupling in $SE(3)$ better than a single pseudo-global state with a Gaussian uncertainty, especially when heading uncertainty is large [30]. The global localization is also improved when the back end is able to optimize the pose graph when it receives other constraints, such as opportunistic GPS measurements and place-recognition loop closures [33, 24]. Graph optimization has been studied
extensively and computationally efficient methods are available [41, 21] for performing these optimizations. Using these techniques, relative navigation deliberately avoids global updates to the front-end filter and thereby increases filter robustness.

The division of the front end and back end also provides additional benefits for scalable UAS operations. First, because the EKF of the front end implicitly draws on the Markov assumption (i.e., the current state and covariance completely represent the previous sequence of events and measurements), it essentially compresses the high-rate sensor information into edges that are published at a low frequency. This compression, effectively pre-marginalization of the graph factors, helps to make the back end scale well for long-duration flights. Also, as the back-end graph grows and the computation of optimization increases, the decoupling of the front end allows the graph optimization to be completed slower than real time if needed, while the front end is still providing full-rate state estimates necessary for vehicle control. Without providing empirical results, [42] hypothesizes that these scalability properties could be beneficial for multi-vehicle cooperative localization.

Prior to this work, the relative-navigation front end has relied on a loosely-coupled VIO where the filter depends on a separate visual odometry algorithm, such as [28], and uses a complete odometry solution as a measurement input. This is depicted in Figure 5.2 with separate boxes for view-based odometry and relative-state estimation. The primary functions of the filter have been to perform the relative-reset step and fuse the odometry with inertial measurements [27]. The keyframe-based visual odometries have been responsible for maintaining visual overlap between the keyframe and the current image by declaring new keyframes regularly and thus have controlled when nodes are declared. They have, so far, resolved scale ambiguity by depending on sensors that measure distance, including laser scanners and RGB-D cameras [33]. Since these sensors are impractical for small, low-cost, fixed-wing UAS, a method that is capable of observing scale without them, such as a visual-inertial odometry, is ideal for this work.
5.2.2 MSCKF

The MSCKF has had a significant impact on the VIO research field. Results have shown that it is capable of maintaining accumulated error less than one percent of the total distance traveled. It has also been proposed for a variety of applications including smart phones [43], ground vehicles [44], and spacecraft [9]. A recent comparison of the MSCKF to other VIO methods [45] shows that its accuracy and consistency performance remains comparable to the state-of-the-art while it is often computationally less expensive.

The work in [46] presented the MSCKF as a dual to EKF SLAM. When EKF SLAM is used as a VIO, the state vector includes states that evolve with the vehicle motion (\(x_{imu}\)) and is augmented with the image feature locations. The state vector \(x\) has the form:

\[
x = [x_{imu}^T f_0 f_1 \ldots f_k]^T
\]

where \(f_k\) is the location of feature \(k\). Although EKF SLAM is relatively intuitive, several issues arise from the fact that the location of the feature is initially unknown by a scale factor and error is introduced when the state vector is augmented. Various modifications have been proposed, including delayed feature initialization [47] and inverse depth parameterization [48], but the addition of initialization error to the state vector with every feature remains an issue with EKF-SLAM-based VIO.

The MSCKF avoids these issues by instead augmenting the state vector with the transformation to the camera at the instant each image is captured. The state vector is therefore defined as

\[
x = [x_{imu}^T \pi_0^T \pi_1^T \ldots \pi_k^T]^T \quad (5.1)
\]

where \(\pi_k\) is the pose of image \(k\). In this formulation, little additional error is added to the state vector during augmentation because the location of the image is well known and its error is cor-
related with error in $x_{imu}$. The state vector contains a time history of image poses that enables feature tracks to be used as measurements given a measurement model [49]. A given feature is tracked across a sequence of images and, once it leaves the camera field of view, the feature track is residualized as a single measurement-update step.

The residual is created by first performing a least-squares optimization to produce the three-dimensional location of the feature given the image poses. The optimized location of the feature is used to produce the predicted pixel coordinates $\hat{z}$ that are subtracted from the measured pixel coordinates $z$ to produce the residual $r$ as

$$ r \triangleq z - \hat{z}. $$

Because the feature location was optimized given both the feature pixel coordinates and the image poses in the state vector, the errors in feature location are correlated with errors in the state vector. This correlation is removed by performing a projection of the residual onto the null space of the feature position. A linear approximation of the residual is produced by two Jacobians: $H_x$ which accounts for the residual with respect to perturbations in the state vector and $H_f$ which approximates the residual with respect to perturbations in the feature location. The residual and Jacobians are fully defined in Appendix 5.9.1. These, with a noise term $\eta$, can be written as

$$ r \simeq H_x \tilde{x} + H_f \tilde{p}_f + \eta $$

where $\tilde{x}$ and $\tilde{p}_f$ are the error in the state vector and position of the feature respectively. The update is then performed by first projecting the residual, noise, and Jacobian $H_x$ onto the null space of $H_f$, or

$$ r_0 \triangleq A^T(z - \hat{z}) \simeq A^T H_x \tilde{x} + A^T \eta $$

34
where $A$ denotes the unitary matrix whose columns form the basis of the left null space of $H_f$. Finally, the projected residual $r_0$ and Jacobian $H_0$ are in an appropriate form

$$r_0 \simeq H_0 \ddot{x} + \eta_0,$$

for use in the Kalman update, given that $H_0 = A^T H x$ and $\eta_0 = A^T \eta$.

The MSCKF has also had several extensions and variations. The original work was extended in the publication of the MSCKF 2.0, which introduces a method for ensuring the state-transition matrix has accurate observability properties [46]. On-line camera calibration, including accounting for rolling-shutter, was introduced in [43] to improve accuracy. Several slightly different formulations of the state vector have been proposed. The work in [44], for example, propagates the estimates using velocity commands and therefore avoids the need for acceleration bias terms. Formulations have used both continuous and fully discrete propagation steps with discrete measurement updates. Finally, to ensure computation remains tractable, several strategies have been proposed for regularly pruning camera poses from the state vector [43, 49].

5.3 Development

The MSCKF measurement model provides a method for constructing a VIO for a fixed-wing UAS because it does not make assumptions about the distance to image features and is both accurate and consistent, at least while nonlinearities due to heading uncertainty remain small. For it to function as a relative-navigation front-end estimator, the original MSCKF must be modified to include a reset step. There is some added complexity and some slight degradation in the filter’s accuracy, compared to the original MSCKF inherent in this approach. The degradation is due to a small amount of information being lost every time a new node frame is declared. We argue that these changes and their benefits, specifically the improved robustness as well as the potential for a light-weight multi-agent back end, outweigh the disadvantages for many applications.
The development of the filter begins by completely defining the state vector in equation (5.1). The pose of the vehicle body \((b)\) consists of a quaternion \(q_b^b\) and a north-east-down position \(p_n^b\) with respect to the most-recent node frame \((n)\). The body of the aircraft is assumed to be centered at and axis aligned with the IMU. In contrast to other MSCKF implementation, the velocity is body-fixed \(v_b\), meaning expressed in the body frame. The complete IMU state is

\[
x_{\text{imu}} \triangleq \begin{bmatrix} p_n^b & q_n^b & v_b & \beta_\omega & \beta_a \end{bmatrix}^T
\]

where the IMU acceleration and angular rate estimated bias are \(\beta_a\) and \(\beta_\omega\) respectively. The transformation to the \(k^{th}\) camera image \(i_k\) are its position and orientation in the node frame,

\[
\pi_k \triangleq \begin{bmatrix} p_i^k & q_i^k \end{bmatrix}^T.
\]  

(5.2)

When an image is taken, these states are calculated from the current IMU state using

\[
p_{n}^{i_k} = p_n^b + R(\mathbf{q}_n^b)p_b^c
\]

\[
q_{n}^{i_k} = q_n^b \otimes q_b^c
\]

where \(\otimes\) is Hamiltonian-quaternion multiplication, \((p_b^c, q_b^c)\) is the calibrated pose of the camera in the body frame, and \(R(q_n^b)\) denotes the rotation matrix associated with \(q_n^b\). It is important to note that the use of the quaternion for rotation requires the filter to use the error-state formulation and be multiplicative. In practice, this means that while the quaternion has four values, to be a minimal representation, the covariance for the same rotation is a three-by-three matrix of the rotational error uncertainty [50, 51].

The covariance \(P\) of the state vector consists of an upper left block that corresponds to the IMU state \(x_{\text{imu}}\). With every image transformation that is added to the state vector the covariance matrix
Figure 5.4: The location of a feature in the node frame $p_n^f$ is obtained through a least-squares optimization, defined in Appendix 5.9.2. The flight path (dotted line) begins at the declaration of a new node frame (red). Also shown are the transformations from the node frame to the keyframe (green), to all other image frames (blue), and to the current aircraft pose.

is augmented as

$$P \leftarrow \begin{bmatrix} P & PJ^T \pi \\ J\pi P & J\pi PJ^T \pi \end{bmatrix},$$

where the Jacobian $J\pi$ relates the current camera location to $x_{imu}$ by accounting for the IMU-to-camera extrinsic parameters. If $[\cdot]$ is the skew-symmetric matrix (defined in Appendix 5.9.1), $J\pi$ is defined as

$$J\pi \triangleq \begin{bmatrix} I_{3,3} & 0_{3,3} & -R^T(q_{n}^b) \mid p_{b}^f \mid 0_{3,3} & 0_{3,3} & 0_{3,6k} \\ 0_{3,3} & 0_{3,3} & R(q_{c}^c) & 0_{3,3} & 0_{3,3} & 0_{3,6k} \end{bmatrix},$$

where $0_{3,3}$ is a 3 by 3 matrix of zeros, and $I_{3,3}$ is the identity. These terms are important because the correlations from the images to the IMU states make the feature track measurements useful in removing error from $x_{imu}$.

The IMU states are propagated with every IMU measurement. The orientation and velocity are mechanized by integrating angular rate and acceleration measurements on the manifold. At each
time step a small amount of process noise is added to the covariance of the bias states to model a slow random walk and to the covariance of the integrated states to model sensor noise. The dynamics are modeled as

\[
\begin{align*}
\dot{p}_n^b &= R^T(q_n^b)v_b \\
\dot{q}_n^b &= \frac{1}{2}q_n^b \otimes \begin{bmatrix} \omega \\ 0 \end{bmatrix} \\
\dot{v}_b &= [v_b] \omega + R(q_n^b)g + a \\
\dot{\beta}_\omega &= \eta_{\beta_\omega} \\
\dot{\beta}_a &= \eta_{\beta_a} \\
\pi_k &= 0_{7 \times 1}
\end{align*}
\]

where \( \eta_{\beta_\omega} \) and \( \eta_{\beta_a} \) are Gaussian noise processes for their respective states, \( g \) is the gravity vector, \( \omega \) is the angular velocity vector, and \( a \) is the acceleration vector. In practice \( \omega \) and \( a \) are obtained by removing their respective bias estimates from the IMU measurements.

The measurement update, including the measurement Jacobians, is formulated to depend on the optimization producing the feature location in the node frame \( p_n^f \). This is in contrast to prior work that has defined the optimizations in the global frame [43] or in the image frame where the feature was first observed [49]. The node frame was used because the filter state is relative to the most recent node and the majority of feature-track measurements are initialized on the keyframe image. An inverse depth parameterization of the feature location is used to perform Levenberg-Marquardt least squares and is defined in Appendix 5.9.2. The coordinate-frame transformations necessary for the optimization are depicted in Figure 5.4.

The relative-reset step consists of removing the heading portion of \( q_n^b \) as well as zeroing the position \( p_n^b \). The uncertainty of the states is also removed by applying a projection to the covari-
The reset step is fully defined in Appendix 5.9.3. In prior relative navigation implementations [33, 24], the reset step was performed after the vehicle had traveled more than a specified distance or yawed more than a specified angle. Since these criteria are insufficient to ensure image overlap, this work makes the criteria for reset depend on the number of feature tracks that are maintained with the most recent keyframe. Once the feature tracker can no longer maintain nine common feature tracks, the reset is performed. This criteria is used to ensure there is sufficient overlap between images and also the number of feature correspondences is adequate to construct a complete transformation between the keyframe and the current image [52]. It has the added benefit of ensuring the reset does not happen sooner than necessary. The state vector is then purged of all image transformations $\pi_k$ and the current image becomes the next keyframe. The state vector is augmented with the keyframe image transformation $\pi_0$ and the keyframe is the first image $i_0$.

5.4 Front-End Implementation

The mathematical development of the filter, while essential, is insufficient without the myriad of implementation details necessary to run and test it. The following sections describe, in part, the simulation implementation details and our efforts to minimize and appropriately account for relevant sources of error that accompany running the filter on hardware.

5.4.1 Filter

The feature tracker implemented a pyramidal KLT tracker [53, 54, 55] using C++ OpenCV libraries. The feature tracker was responsible for informing the filter precisely when to augment the filter state as well as when to perform a reset step. When the feature tracker can no longer track a given feature, e.g. if the feature goes out of view, the tracker provides to the filter the complete track as a measurement, consisting of the history of pixel coordinates for every image where it was observed.
Although it was initially developed in Python [40], the filter was implemented in C++ and uses the Robot Operating System (ROS) for communication with sensors. The C++ implementation allowed the filter to run in real time and at full sensor rate, even on an embedded ARM processor.

### 5.4.2 Simulation

The filter was first tested in a ROS/Gazebo simulation using the tools that are distributed with ROSplane [19, 18]. The fidelity of the simulation was enhanced by simulating a small fixed-wing aircraft, including aircraft aerodynamics, flight characteristics, and sensors. The aircraft was flown in a realistic flight over a cityscape image appropriate for obtaining image features and testing a VIO algorithm.

In the simulation, sensor plug-ins were used to supply the filter with simulated camera images and IMU measurements from the aircraft. The IMU was oriented to be axis-aligned with the body of the aircraft and noise and bias walk parameters where representative of an MPU-6050 IMU, based on models presented in [56, 57]. Feature tracks were obtained from the simulated camera image using the tracker described previously. An example of the simulated image and image feature tracks is shown in Figure 5.5. The camera was oriented facing forward and tilted 45°.
degrees down from the longitudinal axis of the aircraft. The images were 640 by 480 pixels and the camera had a 115 degree field of view.

5.4.3 Hardware

The front-end filter was implemented on a small remotely-piloted hobby-grade aircraft, Figure 5.1. The aircraft carried an NVIDIA Jetson TX2 embedded computer. The use of the OpenCV CUDA functionality was utilized to perform image processing on the GPU. The use of the GPU freed the CPU to perform other tasks and reduced the tracker CPU load from 130% to 30% of a single processor core. The TX2 received images from a Point Grey Chameleon 3 USB camera and the acceleration and angular-rate gyroscope measurements from a thermally calibrated InertialSense IMU. This IMU is also a micro GPS-INS and is capable of producing a full navigation solution for truth comparison.

The hardware implementation introduced three sources of error that were not initially considered in the simulation: calibration error, timing error, and initialization error. Without addressing these errors, the filter would either diverge or give unsatisfactory performance. Figure 5.6 demonstrates the sensitivity of the estimator accuracy to these types of errors when they are deliberately introduced into the simulation without accounting for them as described in the following sections.

5.4.3.1 Calibration Error

Initial testing showed that a satisfactory calibration of the camera’s intrinsic parameters can be performed prior to the flight. Error in the extrinsic parameters, specifically the body (IMU) to camera rotation angles, however, was detrimental to the filter performance. Since the transformation from the body to camera is used in the measurement-model calculation of the residual \( r \) and measurement Jacobians \( H_x \) and \( H_f \) (see Appendix 5.9.1), the error is correlated with every feature measurement and it causes significant bias in the estimates. As shown in the left plot in Figure 5.6, position error increases as angular errors are added to the body-to-camera offset. With 0.8 degrees
of error or less, the RMS error appears to increase quadratically with calibration error. When the calibration error is greater than approximately 0.8 degrees the outlier rejection within the filter begins to prevent measurements from negatively affecting the estimates and RMS error continues to increase approximately linear with additional calibration error.

This calibration error was accounted for, and removed in flight, by introducing the camera rotation to the state vector, making Equation (5.1) become

$$x = [x_{imu}^T \ q_b^c \ T \ \pi_0^T \ \pi_1^T \ \ldots \ \pi_k^T]^T$$

where \( q_b^c \) is a quaternion of the rotation from the body (IMU) frame to camera frame. The covariance was initialized to a relatively large value and allowed to converge over time. We note that the introduction of \( q_b^c \) to the state vector makes the use of \( q_{ik}^d \) in Equation (5.2) a non-minimum representation of the state because the camera pose includes the calibrated camera rotation. This slight mis-modeling, and the fact that the camera calibration is both static and minimally observable in this case (monocular camera and unknown features), necessitates the use of partial updates [58] (defined in Appendix 5.9.4) to avoid inaccurate updates to \( q_b^c \) during in-flight calibration. The body-to-camera position offset \( p_b^c \) was not included in the online calibration for two reasons. First,
the physical distance was small (about 2 cm) compared to the baseline to observed features, making it nearly unobservable and, second, testing using the simulation environment showed the filter performance was insensitive to errors in $p^c_b$.

In general, the mathematical development provided in this paper, including the appendices, corresponds to the original state vector Equation (5.1). The inclusion of $q^c_b$ as an estimate introduces only minor modifications and similar efforts are discussed in [43].

### 5.4.3.2 Timing Error

The center plot in Figure 5.6 demonstrates that as little as 10 ms of timing error in the sensor measurements is enough to approximately double the RMS position error, and when the timing error was 20 ms or more, the timing error produced a corresponding bias in the direction of travel. Since the TX2 was not running a real-time operating system, the filter depended on accurate time stamps on each measurement. The image measurements were prevented from receiving an accurate time stamp by significant and varying delays introduced while transferring the images from the camera to the computer. To overcome this delay, the camera was configured to provide a strobe pulse that coincided with the camera shutter. Each pulse caused the InertialSense IMU to publish the current time stamp. Once the time stamps were received on the TX2 computer they were added to a queue and used to re-stamp the images once they were fully transferred from the camera. Since both the IMU measurements and the GPS-INS, truth navigation solution also originated from the InertialSense IMU, every necessary measurement was stamped relative to the same time reference. Because recombining time stamps with their corresponding images depended only on their order in the queue, there was some ambiguity in their association making this method only reliable up to an image frame rate of 7 Hz.

Once the measurements were stamped with the correct time, they were used by the filter even though the images (and thus the feature track measurements) arrived after the IMU measurements. The filter uses the out-of-order measurement scheme described in [33], where sensor measurements
and filter state snapshots are stored in a priority queue. When an old measurement arrives, the filter rewinds to just before the new measurement, applies it, and then fast-forwards (and updates the snapshots) to the latest measurement. Because image measurements incur more delay and IMU measurements only propagate the IMU state $x_{imu}$, this method is computationally feasible and runs in real time.

5.4.3.3 Initialization Error

The MSCKF measurement model, including augmenting the state vector with the time-history of image transformation, performs well once the filter has converged to the true value, but suffers when there are significant errors in the IMU state. This is particularly problematic during initialization. Assuming the aircraft is not moving when the filter starts, position and velocity can be initialized to zero with negligible covariance and the angular-rate bias can be determined from the first few measurements. The filter must be initialized, however, to an unknown attitude $q_n^b$ and acceleration bias $\beta_a$. These states cannot easily be sensed by measuring the gravity vector because attitude errors and acceleration bias are correlated. The right plot in Figure 5.6 shows how filter performance is sensitive to initial roll and pitch attitude error. In the simulation tests, the initial covariance for the attitude states did not accurately model the error. If the initialization errors are small and the flight is sufficiently long, the initialization errors often die out as the estimates converge, but not until after the odometry has accumulated error that can significantly reduce the accuracy of the navigation solution.

Our strategy for initialization of the filter (in the hardware flight tests only) included using the InertialSense GPS-INS attitude to initialize the filter attitude and using its reported body-frame velocity as a measurement for the first 45 seconds to help the acceleration bias states converge. Using the velocity as a measurement was advantageous because the relative-reset step did not affect how the measurements were utilized by the filter. Conversely, using the reported position would have required transforming the measurement into the node frame using a potentially inaccurate
Figure 5.7: Top: Three simulation flight tests where the true path (blue) is compared to the accumulated estimate (red). Bottom: The error as a percent of the distance traveled is shown for the first 60 seconds of each flight. There are significant bias errors when the aircraft flies straight and level due to velocity being less observable for a monocular VIO. The estimates improve significantly when a non-straight trajectory is used, even with a slight sinusoidal s-turn. In the flight with the most deviation from straight the accumulated error is ultimately less than 1% of the distance traveled.

attitude estimate. The use of the partial update [58] on acceleration bias and on attitude states improved the consistency of the filter and limited its confidence of the estimates. The partial update is detailed in Appendix 5.9.4.

5.5 Front-End Results

The filter was first tested in the high-fidelity fixed-wing simulation described above. Because the filter publishes the position relative to the most recent reset-step node frame, to plot and analyze the performance of the filter, the state must be put into the global frame or the truth must be put in the node frame. In Figure 5.7 the estimates are put into the global frame by composing current state with the previously published edge transforms, similar to a back-end graph. Figure 5.7 compares the front-end results for 60 seconds of three different simulated trajectories, and shows the results suffer when the aircraft flies straight and level, but improve as the aircraft turns [59]. This
Figure 5.8: Left: The aircraft flight path during a manually-flown flight test. Right: The true path (blue) is compared to the accumulated estimate (red). The estimates from 150 to 350 seconds (green) are also shown to compare the result of removing most of the effects of the initialization errors. Other than the take-off and landing circles, the aircraft was flown approximately straight and level over a 6 km distance. Notice the scale bias in the estimates.

Figure 5.9: The true (blue) and estimate (red) states of a UAS in flight relative to the most recent node. Position, velocity, and attitude states are shown from left to right respectively. The relative reset associated with the declaration of a new nodes are shown with gray vertical lines.
Figure 5.10: State error (blue) is shown for position, velocity, and attitude states (from left to right respectively). The $3\sigma$ uncertainty bounds (red) come from the square root of the diagonal terms of the covariance matrix. Sharp decreases in the position error bounds are due to the relative-reset step that are also indicated by gray vertical lines.

phenomenon is particularly important for fixed-wing aircraft because they often fly over greater distances to accomplish mission objectives. For the trajectory with the most turns the total accumulated error is shown as less than 1% of the distance traveled, where the distance traveled is the integrated flight-path length.

Hardware flight results were also obtained by flying the aircraft in Figure 5.1 over a 6 km trajectory. The front-end estimates were produced on the aircraft in real time. The true flight trajectory and accumulated estimates are shown in Figure 5.8. Because the aircraft was flown by a remote pilot, the trajectory is only roughly straight and level, that is, other than during take-off and landing. The effects of the initialization error can also be seen in the first 100 seconds of the flight when the scale error is much greater. The filter estimates from 150 to 350 seconds are also shown to compare the performance after the estimates converge and the effects of initialization are minimized. The total accumulated error of the filter estimate from 150 to 350 seconds was approximately 2.5 percent of the distance traveled. The entire data set was 388 seconds long and the total accumulated error for the flight, corresponding to the red trajectory in Figure 5.8, was 5.3 percent of the distance traveled.
The results in Figure 5.9 show the estimates track the true motion of the aircraft. The effect of the relative reset can be seen when position and heading angle abruptly return to zero, as previously defined. During a reset step a new origin is declared at the position of the aircraft, ensuring both the true and estimated values return to zero. The estimate velocity and roll and pitch angles do not reset. The amount of time between resets varies depending on there being more than nine continuous feature tracks but generally is between 1 and 7 seconds.

Figure 5.10 shows $3\sigma$ bounds around the relative error. The bounds are calculated from the square root of the diagonal terms of the covariance matrix $P$. From these plots it appears the filter is consistent and the uncertainty grows approximately linearly with the distance traveled. The effect of the relative-reset step can be seen as the error and $3\sigma$ bound both return to zero for the position and heading states. The filter also publishes its position and heading state (and associated covariance) from just prior to the reset to be used in a relative-navigation, back-end pose graph.

5.6 Back End

In all prior relative-navigation work, the global back-end graph optimization has assumed that the edges published from the front end have been statistically independent, meaning errors in one edge were uncorrelated with errors in all others. This assumption has worked well when the errors in the estimated linear velocities remained small due to direct depth measurements [24] or flying with s-turns to help velocity remain observable [40]. The assumption becomes less appropriate, however, when errors are more significant. In this paper, velocity error is more significant for a fixed-wing UAS flying straight and level over extended periods causing the forward velocity to be less observable [59], and therefore the edge errors to be more correlated. Because feature tracks are discarded and reinitialized at each keyframe and its associated relative-reset step, the velocity estimates are likely degraded when compared to the nominal MSCKF. It is our belief that any disadvantages introduced by the relative reset are mitigated or effectively eliminated by the
Figure 5.11: Factor graphs used in the global back-end where values are ovals and measurement factors are squares. Top: Original graph where nodes ($N$) are connected by odometry edge factors ($E$) from the front-end filter and the edges are modeled as independent. GPS measurements or other global constraints can be opportunistically added as unary factors. Bottom: In our method edges become a trinary factor which also considers a bias scale variable ($B$). Because velocity errors persist through a relative-reset step and errors in the edges are correlated, the bias is modeled as a random walk through the use of binary factors ($R$) which are initialized as identity with small, non-zero covariance.

Optimizations performed in the back-end. This is especially true when the back end incorporates intermittent global measurements or other loop-closure-like constraints.

Velocity errors can be accounted for over a single edge by applying a scale bias to the published position. Because velocity errors persist through a relative-reset step, they are also correlated between consecutive edges. This correlation is similar to how gyro bias walks and is correlated over time.

In the back-end we model a two-dimensional slowly-varying bias walk using trinary factors for edges ($E_k$), which are similar to IMU preintegration factors that account for IMU bias [60]. Each node variable $N$ includes the global north and east position ($p = [n e]^T$) and global heading ($\psi$) and each bias variable $B$ includes the scale bias ($b = [b_x b_y]^T$). The factor is then defined by a loss function $\ell$ that effectively rotates the change in global position into the previous node frame,
applies the bias scale, and then subtracts the measured odometry. The function is defined as

$$\ell(N_k, N_{k+1}, B_k, m) = \begin{bmatrix}
    (\cos(\psi_k)(n_{k+1} - n_k) + \sin(\psi_k)(e_{k+1} - e_k))b_x - m_x \\
    (-\sin(\psi_k)(n_{k+1} - n_k) + \cos(\psi_k)(e_{k+1} - e_k))b_y - m_y \\
    (\psi_{k+1} - \psi_k) - m_\psi
\end{bmatrix}$$

where $m$ is the measurement of the edge odometry published by the front end at each relative reset and includes the change in position ($m_x$ and $m_y$) and heading ($m_\psi$). The factor graph is shown in Figure 5.11 with the same depiction style as in [41] and is implemented using GTSAM.

Unlike gyrocope bias, velocity errors are not a stochastic process. The autocorrelation of the velocity errors depends on the observability of the velocity and thus the flight trajectory of the aircraft, and the scale error correlation between edges depends on the time between resets. This means modeling the bias scale as a random walk is a simplification. In practice, the covariance of the binary factor ($R$) between bias variables ($B_k$) is hand tuned. In the results shown below these factors use $\Sigma = 0.0001I_{2 \times 2}$, where $I_{2 \times 2}$ is a 2 by 2 identity. This extension may be most relevant for fixed-wing aircraft using VIO but could also be applied to all previous relative-navigation work where velocity errors persist through the reset step.

Once the factors are defined, they can be added to the graph with connections to the appropriate variables. The variables in the graph are the global north, east, and yaw poses of the keyframe nodes ($N$) and the bias ($B$) at each odometry and are initialized appropriately. Finally, GTSAM provides functions to optimize the graph such that the loss of all the combined factors is minimized. The resulting graph, and thus the global state, are produced by optimizing after all the factors have been added. The full details of factor-graph optimization are extensive but can be ascertained from [41], [21], [60], and elsewhere.
Figure 5.12: Back-end optimization of the flight-test graph. The unoptimized trajectory (red) is the raw front-end estimates in a graph but before optimization (corresponding with the red trajectory in Figure 5.8). Three simulated GPS measurements were added to help with initialization. Scale-bias factors were used to remove the scale error of the estimated edges. The blue background indicates areas where GPS was available.

5.7 Full System Results

To demonstrate the value of the proposed relative front end, the full localization solution is produced in a single back-end graph using the published edges from pre-recorded hardware flight tests. The results shown in this section utilize a two-dimensional graph that is optimized post-process and in a single batch, although similar back-end architectures have been shown to work on single multirotor aircraft for both localization and navigation in near real time [24]. The back-end results simulated global measurements calculated from the reported states from the InertialSense GPS-INS that were used for truth comparison.

Figure 5.12 incorporates three simulated GPS measurements into the graph. The GPS measurements were added to the graph with a 0.32 m standard deviation error. These measurements help remove initialization errors and provides constraints to optimize the scale factors introduced previously. The results represent a mission profile where GPS is available until the aircraft enters an area where the GPS is spoofed or jammed or otherwise unavailable.
Figure 5.13: Back-end optimization of the flight-test graph includes five simulated range measurements to two static features or DME stations. The results improve significantly by removing initialization and scale errors.

The results in Figure 5.13 also incorporate simulated global measurements, but in this example the back end utilizes five distance measurements to two static features. The range measurements represent measurements to a distance-measuring-equipment (DME) transponder or similar fixed ground-based range station. The range measurements were simulated as having a 0.71 m standard deviation error. These results again show the ability of the back end to improve global accuracy. If a given range measurement had been used as an update to the EKF, however, it may have caused the filter to become inconsistent or even diverge, depending on the amount of uncertainty. By incorporating these inputs in the back-end and not in the filter, this approach avoids the worst case scenario while still improving the localization.

Finally, the results in Figure 5.14 use relative inter-vehicle range measurements (with a simulated 0.71 m standard deviation) between aircraft flying in a small formation rather than from a stationary ground feature only. The measurements allow the aircraft to cooperatively localize. The three trajectories depicted are from separate flight tests of the test aircraft in Figure 5.1. Each trajectory includes significant initialization errors with bias scale. The center aircraft receives intermittent, simulated global position measurements such as those from GPS or a computationally
expensive satellite-image-based place recognition system [9]. The results show that not only do the outside aircraft receive the benefit of the global measurements but also the relative position of the formation is maintained.

These later results do not account for several aspects of a full multi-vehicle cooperative solution, including the necessary communication links between vehicles. They do show that the proposed method holds promise for these scenarios. For example, a rough estimate of the total amount of sensor data processed is 5.8 GB for all three vehicles, whereas the back-end graph is constructed from less than 0.15 MB of data. This suggests the potential for both the scalability of a multi-agent system as well as robustness to communication loss or delay.

5.8 Conclusion

This paper has demonstrated a method for localizing a fixed-wing UAS in environments where GPS is either unavailable or unreliable. This work has used the relative navigation architecture, which was previously implemented for multirotor UAS, as a guide. The front-end filter depends on
a camera and an IMU for sensing and has no other specific requirements. It uses a VIO approach to estimate the motion of the aircraft and regularly publish transformations that can be used in a back-end graph. The filter uses a modified MSCKF measurement model and the relative-reset step. The filter also makes no assumptions about the scale or distance to observed image features.

The filter was tested first in simulation. The simulation testing showed the filter accuracy is trajectory-dependent due to the lack of observability of the velocity in straight-and-level flight. In simulation the total accumulated error is demonstrated as less than one percent of the distance traveled, provided there is sufficient turning to maintain observability of the velocity estimate. The front-end filter was also demonstrated in a hardware flight test. The implementation details of the flight test, including our efforts to account for calibration, timing, and initialization errors, were discussed. After the initialization errors are removed the filter was accurate and ultimately accumulated error approximately 2.5 percent of the distance traveled.

The value of this approach can best be evaluated by considering the whole relative-navigation architecture. The estimates from the relative-navigation, front-end, VIO estimator are used in a back-end graph. The back-end graph is responsible for both representing and optimizing the global state, which is necessary for accomplishing a global mission. For the back end to more accurately utilize our front-end estimates, we introduced a scale-bias model to account for the correlation of the scale errors between edges.

This work has also demonstrated the use of the back end and graph optimization to incorporate other constraints, such as opportunistic, geo-referenced measurements. Such measurements can be problematic for a front-end EKF because large covariance and filter inconsistency can cause the update to produce large jumps in the state. Because the proposed relative front-end operates independently from the back end, jumps in the back-end state do not directly affect the control of the vehicle. The full system is able to operate over significant periods without global information
and whenever it becomes available the system can seamlessly utilize it in the back end to improve localization.

Finally, we have shown the potential for the proposed method and the relative-navigation architecture to be used in multi-vehicle cooperative localization scenarios. The back-end graph is able to efficiently incorporate the odometry edges from multiple vehicles as well as relative inter-vehicle and global measurements. Cooperative, multi-vehicle localization will be explored in future work.

Acknowledgment

This work has been funded by the Center for Unmanned Aircraft Systems (C-UAS), a National Science Foundation Industry University Cooperative Research Center (IUCRC) under NSF award No. IIP-1650547 along with significant contributions from the AFRL Munitions Directorate and other C-UAS industry members. Gary Ellingson also received support from the Utah NASA Space Grant Consortium Fellowship.

5.9 Appendix

5.9.1 Measurement Jacobians

This section defines the measurement Jacobians $H_x$ and $H_f$ that are necessary for the MSCKF measurement model. As described previously, the measurement is the pixel coordinates of a feature track. Thus we begin by providing a camera projection function $h(p)$ to project a feature in the image frame onto a pixel coordinate while accounting for the camera matrix and distortion parameters. The projection function enables the construction of the predicted measurement and residual. Finally we provide the partial derivatives to fully define the Jacobians.
If we first neglect distortion, the camera projection function consists of normalizing the feature position vector (in the image frame) $p_f^i$ by the depth and multiplying it by the camera matrix $K$ or

$$h(p_f^i) = K \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

(5.3)

where

$$p_f^i \triangleq \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix},$$

$$u \triangleq \frac{p_x}{p_z},$$

and

$$v \triangleq \frac{p_y}{p_z}.$$ 

Next a camera distortion model that includes radial coefficients ($k_1, k_2, k_3$) and tangential coefficients ($t_1$ and $t_2$) is applied to $u$ and $v$ using

$$r = u^2 + v^2$$

$$d_r = (1 + k_1 r + k_2 r^2 + k_3 r^3)$$

$$u' = d_r u + 2 u v t_1 + (r + 2 u^2) t_2$$

$$v' = d_r v + 2 u v t_2 + (r + 2 v^2) t_1.$$ 

Finally the projection function $h(p_c^i)$ consists of substituting $u'$ and $v'$ for $u$ and $v$ respectively into Equation (5.3).

Recall that the measurement $z_k$ is the pixel location provided by the tracker for camera image $i_k$ and the least-squares optimization produces $p_f^n$, the position of the feature in the node frame. The
residual $r$ can then be constructed by transforming the feature position to the appropriate camera image frame with

$$p_{ik}^f = R_{n}^{i_k}[p_{n}^f - p_{n}^{i_k}]$$

and then projecting it using the projection function $h(p_c^f)$. Thus the residual for a single feature tracked over several images is

$$r = \begin{bmatrix}
    z_0 - h(R_{n}^{i_0}[p_{n}^f - p_{n}^{i_0}]) \\
    z_1 - h(R_{n}^{i_1}[p_{n}^f - p_{n}^{i_1}]) \\
    \vdots \\
    z_k - h(R_{n}^{i_k}[p_{n}^f - p_{n}^{i_k}])
\end{bmatrix}.$$  

Finally, in constructing the measurement Jacobians we define the partial derivative of the camera projection function as

$$J_k \triangleq \frac{\partial h(p)}{\partial p},$$

the skew-symmetric matrix for a vector as

$$\lfloor a \rfloor = \begin{bmatrix}
    0 & -a_z & a_y \\
    a_z & 0 & -a_x \\
    -a_y & a_x & 0
\end{bmatrix},$$

and the partial derivative of the residual with respect to the image transformation $\pi_k$ as

$$H_{\pi_k} \triangleq \frac{\partial r}{\partial \pi_k} = \begin{bmatrix}
    -J_k R_{n}^{i_k} & J_k \left[ R_{n}^{i_k}[p_{n}^f - p_{n}^{i_k}] \right]
\end{bmatrix}.$$
The measurement Jacobians are

\[
H_x = \begin{bmatrix}
0_{2 \times 15} & H_{\pi_0} & 0_{2 \times 6} & \cdots & 0_{2 \times 6} \\
0_{2 \times 15} & 0_{2 \times 6} & H_{\pi_1} & \cdots & 0_{2 \times 6} \\
\vdots & \vdots & \ddots & \ddots & \ddots \\
0_{2 \times 15} & 0_{2 \times 6} & 0_{2 \times 6} & \cdots & H_{\pi_k}
\end{bmatrix}
\]

and

\[
H_f = \begin{bmatrix}
J_0 R_{i_0}^n \\
J_1 R_{i_1}^n \\
\vdots \\
J_k R_{i_k}^n
\end{bmatrix}.
\]

In practice several tracks can be used in a single update by vertically stacking residuals \( r \) and the measurement Jacobians \( H_x \) and \( H_f \).

### 5.9.2 Feature Optimization

As part of the measurement model described previously, a least-squares optimization is performed to produce the position of the feature in the node frame (\( p_{f_n}^i \)) using the image transformations (\( \pi \)) and the pixel-coordinate measurements (\( z \)). This section defines the Levenberg-Marquardt least-squares optimization that is depicted in Figure 5.4.

For numerical stability, we optimize an inverse-depth parameterization of the feature position \( \rho = [p_x \ p_y \ -1 \ p_z]^T \). Next, we define a function \( g \) that receives \( \rho \) in the node frame and the camera image pose and produces the feature position transformed into the image frame, or

\[
p_{f_k}^i = g(\rho_{n}^f, \pi_k) \triangleq R(q_{n}^i) \begin{bmatrix} u \\ v \\ \frac{1}{p_z} \end{bmatrix} - \frac{1}{p_z} R(q_{n}^i) p_{n}^i.
\]
The position of the feature can be projected into pixel coordinates of the image using the camera projection matrix and distortion parameters defined in Equation (5.3) of the previous section.

We now setup the formal optimization problem

\[
\min_{\rho} f(\rho_n^f) = \sum_{k=0}^{n} \left[ z_k - h(g(\rho_n^f, \pi_k)) \right]^2.
\]

Finally, after the optimization is completed the position of the feature \( p_n^f \) is extracted from \( \rho_n^f \).

### 5.9.3 Reset Step

This section describes the process for performing a relative-reset step. The relative-reset step is performed as the position and heading state estimates and their uncertainties are removed from the front-end filter and a new local origin is declared. We discuss first removing the estimate and then the uncertainty from the covariance matrix.

Removing the position from the state vector is performed by simply applying zeros to the position vector or

\[
p_n^b \leftarrow 0_{3\times1}.
\]

The orientation of the body in the node frame is represented by a quaternion \( q_n^b \). Removing the heading from the quaternion is non-intuitive, however, and we instead decompose it to Euler angles, remove the heading and finally reconstruct the quaternion. Using the common aircraft attitude representation of roll \( \phi \), pitch \( \theta \), and yaw \( \psi \) as the active 3-2-1 Euler angles and a Hamiltonian
quaternion, the decomposition is

\[ \phi = \text{atan} \left( \frac{2q_0q_x + 2q_yq_z}{q_x^2 - q_y^2 - q_z^2 + q_0^2} \right), \]

\[ \theta = \text{asin} (2q_0q_y - 2q_xq_z), \]

\[ \psi = \text{atan} \left( \frac{2q_0q_z + 2q_xq_y}{q_x^2 - q_y^2 - q_z^2 + q_0^2} \right). \]

The new quaternion is constructed from the roll and pitch angles and zero for yaw (\( \psi = 0 \)) by applying equations

\[ q_x = \cos \frac{\psi}{2} \cos \frac{\theta}{2} \sin \frac{\phi}{2} - \sin \frac{\psi}{2} \sin \frac{\theta}{2} \cos \frac{\phi}{2}, \]

\[ q_y = \cos \frac{\psi}{2} \sin \frac{\theta}{2} \cos \frac{\phi}{2} + \sin \frac{\psi}{2} \cos \frac{\theta}{2} \sin \frac{\phi}{2}, \]

\[ q_z = \sin \frac{\psi}{2} \cos \frac{\theta}{2} \cos \frac{\phi}{2} - \cos \frac{\psi}{2} \sin \frac{\theta}{2} \sin \frac{\phi}{2}, \]

\[ q_0 = \cos \frac{\psi}{2} \cos \frac{\theta}{2} \sin \frac{\phi}{2} + \sin \frac{\psi}{2} \sin \frac{\theta}{2} \cos \frac{\phi}{2}. \]

When removing the uncertainty from the covariance matrix \( P \) we only consider the IMU portion of the state or \( x_{imu} \), as the augmented camera image transforms \( \pi_k \) are removed from the state vector during the reset. This is done by constructing a projection matrix \( N \) and applying it to \( P \) using

\[ P \leftarrow NP_{15 \times 15}N^T \]

where

\[
N \triangleq \begin{bmatrix}
0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 6} \\
0_{3 \times 3} & I_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 6} \\
0_{3 \times 3} & 0_{3 \times 3} & N_g & 0_{3 \times 6} \\
0_{6 \times 3} & 0_{6 \times 3} & 0_{6 \times 3} & I_{6 \times 6}
\end{bmatrix}
\]
and

\[
N_q \triangleq \begin{bmatrix}
1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\
0 & \cos^2 \phi & -\cos \phi \sin \phi \\
0 & -\cos \phi \sin \phi & \sin^2 \phi
\end{bmatrix}.
\]

### 5.9.4 Partial Update

The partial-update Schmidt-Kalman filter (PSKF) was introduced in [58]. The PSKF generalizes the classic EKF update step and offers a simple and effective approach to improve the EKF’s consistency and robustness when estimating problematic and mildly-observable filter states. It is an extension of the core concept behind the Schmidt-Kalman filter [61] resulting in the ability to reweight the classic filter update to apply anywhere from 0 to 100 percent of the nominal EKF update for each state at each update step.

Unlike a Schmidt-Kalman filter, which applies a zero update to so-called *nuisance* states and full updates to all other states, the partial updates can be applied both to static nuisance states, as well as classic *full* states. The partial update is performed by first calculating the full Kalman update using

\[
K = P^- H^T (HP^- H^T + R)^{-1}
\]

\[
\hat{x}^+ = \hat{x}^- + K(r)
\]

\[
P^+ = (I - KH)P^-.
\]

The state and covariance is then partially updated with

\[
\hat{x}_i \leftarrow \gamma_i \hat{x}_i^- + (1 - \gamma_i) \hat{x}_i^+
\]

\[
P_{ij} \leftarrow \gamma_i \gamma_j P_{ij}^- + (1 - \gamma_i \gamma_j) P_{ij}^+
\]
where \( \gamma_i \) is from a user defined \( \gamma = [\gamma_0 \cdots \gamma_n]^T \) and chosen such that \( \gamma_i \in [0, 1] \). The value \( 1 - \gamma \) can be thought of as the percentage of the full update applied to state \( i \). For example, \( \gamma_i = 0 \) implies the full EKF update is applied to state \( i \) while \( \gamma_i = 1 \) implies that state is simply considered, anything in between would result in a partial update of the state. Generally less observable, slowly time-varying states should receive a lower percentage of the full update, while more observable states with higher process noise or uncertainty growth rates would receive larger (or full) updates.

The partial-update approach was shown to increase filter robustness to large uncertainties in camera to IMU calibration example in [58]. In our filter, the partial update is applied on the acceleration bias \( \beta_a \), body attitude \( q_b^b \), and camera to IMU rotation \( q_b^c \) states. The results were obtained with \( \gamma \) values of 0.9, 0.9, and 0.97 respectively (and 0 for all other filter states), implying that 10%, 10%, and 3% (and 100%) of the nominal updates were applied to the respective states at each measurement update step.
CHAPTER 6. COOPERATIVE RELATIVE NAVIGATION OF MULTIPLE AIRCRAFT IN GPS-DENIED/DEGRADED ENVIRONMENTS

6.1 Introduction

For unmanned aircraft systems (UAS) to be utilized fully in commercial and military applications, they must be resilient to intermittent or degraded GPS, including complete loss of GPS for significant intervals. Although UAS have gained advanced navigation capabilities through the optimization and miniaturization of traditional navigation methods that fuse GPS and inertial measurements (GPS-INS), they need to maintain their navigation abilities when the GPS signal is degraded or unavailable.

Almost universally, UAS use inertial sensors to estimate their motion with respect to an inertial reference frame [62]. When position measurements, such as measurements provided by GPS, are unavailable the global position and yaw angle are unobservable [1, 2, 3], measurement errors accumulate, and the state estimates drift from the true values. To operate in GPS-denied environments, UAS often include exteroceptive sensors, such as cameras or laser scanners, to measure the vehicle motion relative to the surroundings. By including the additional measurements, the drift rate of the estimated states is reduced. The accuracy of vehicle position estimates can be further improved if multiple GPS-denied vehicles can collaboratively share and combine information about their motion.

Another way UAS can benefit from cooperative, GPS-denied navigation is by coordinating to achieve a common goal. There are many scenarios where a given UAS may be more concerned with its position relative to an objective, a target, or another UAS than its global position. It has

\[1\] Submitted to AIAA Journal of Aerospace Information Systems
been shown that UAS can more accurately sense an objective and achieve a goal by coordinating with other UAS and sharing measurements [63]. As an example, consider a group of independent UAS that have a mission objective of intercepting an opponent aircraft with limited inter-vehicle communication. Some of the aircraft have their own measurements to the opponent’s position while others rely on the measurements provided by their companions. In this example the relative position between each agent and the target is essential to performing the task and multi-vehicle cooperative navigation would be beneficial. The scenario objective can be achieved, however, with or without accurate knowledge of global position and/or access to GPS measurements. Several similar examples of the benefits of coordination are discussed in [42].

There are several significant challenges in allowing UAS to cooperatively navigate or localize. First, a potential solution must define what information is shared. A naive approach would be to share sensor measurements, but the associated high data rate could be problematic for communication networks, especially as the number of vehicles increases. Next, the method of fusing the information needs to be determined. Filtering methods, including extended Kalman filters (EKFs), are computationally efficient for data fusion but are brittle to delays in communication. Graph optimization methods are able to improve a pose graph by incorporating sensor measurements as factors. Graph methods do not utilize a Markov assumption and, therefore, are able to incorporate delayed information when it becomes available. Optimizing a graph can become computationally expensive as the number of factors increases. Further, cooperative approaches can either be centralized or decentralized. Often, resource constrained UAS rely on a centralized computer to perform the computationally intense data fusion methods and optimizations. Centralized approaches, however, make the agents dependant on the central computer and they therefore must maintain communications. Though solutions exist that address various combinations of these challenges, they often make strict assumptions that can limit the specific approach to fast, high-bandwidth, or uninterrupted communication.
This work builds on a previously developed concept called relative navigation as the overarching framework for GPS-denied operations. The relative navigation architecture is divided into a front end that is responsible for real-time estimation and control of the UAS with respect to the local surroundings, and a back-end that performs a graph optimization to improve global navigation. This work is a direct follow-on to [64] by utilizing the front-end, fixed-wing, key-frame-based estimator that was presented previously in [40] and modifies it to include criteria for resetting the estimator based on coordination between aircraft. This work enables cooperative-localization during flight by allowing each UAS to include front-end odometry from other UAS in their back-end graphs.

This paper begins by reviewing relevant research, including cooperative navigation of UAS and the relative navigation architecture. We then provide a modification to the relative navigation framework to enable coordination between multiple vehicles and to define a simple communication protocol. The system is demonstrated using two examples. First, simulation results are presented to explore and quantify the benefits of cooperative localization. Second, results from a hardware flight demonstration show the performance of the complete system. Finally, results from post-processing the flight data are used to analyze the communication constraints and demonstrate global constraints in the back-end graph. The method is demonstrated as a fully decentralized approach that is scalable and robust to limited, unreliable, low-bandwidth communication. The method is also opportunistic, meaning a single agent can benefit from cooperative localization when possible but may operate without relying on it.

6.2 Related Works

The literature on cooperative UAS operations is extensive. Most early work investigated cooperative planning to coordinate a task [65]. Often planning is performed with a centralized approach and in some instances it can be decentralized by applying a consensus algorithm [66]. One of the most publicly visible applications of cooperative technology is Intel Corporation’s Shooting Star
program, which has flown over 2000 drones simultaneously [67]. Nearly all of this work assumes the existence of highly accurate state estimates, usually relying on GPS with real-time-kinematic (RTK) updates to improve estimates. Communication and networking has also been studied for UAS swarms [68] and for other purposes using UAS as relays [69].

The quantity of relevant research on cooperative localization of UAS, especially in GPS-denied environments, is markedly reduced. There are a couple of approaches that focus on using vision, and sharing images, to find overlap in field of view between agents [70, 71]. The work in [72] compares architectures for performing simultaneous localization and mapping in several UAS by sharing map feature measurements. The works [11, 12] use a centralized EKF for cooperative localization. Large initial uncertainty is overcome through multi-hypothesis testing in [13]. Range-only measurements between vehicles has been demonstrated to be useful in GPS-denied environments for fixed-wing and ground vehicles [13, 73]. The number of cooperative approaches that focus on ground vehicles is also extensive [22].

This work is perhaps most similar to [23], but includes the following important differences. First, that work assumed full, high-bandwidth connectivity between robots for communication, an assumption that we avoid. In fact, we explicitly address a limited communication scheme, that performs well even with extended communication dropouts. Next, the results of [23] utilized slow-moving ground and multi-rotor vehicles that moved in-and-around structured environments and used laser-scanners to measure the motion of the vehicles. While this work focuses efforts on fixed-wing UAS that fly high above the environment where depth sensors are impractical [64]. Finally, in [23] inter-vehicle measurements were based on vision and relatively-large, high-contrast, fiducial markers, which are impractical for many real-world scenarios.

The communication scheme presented in [74] has similarities to what is proposed below but uses high-bandwidth WiFi connections. That work showed that the decentralized solution is approximately equivalent to a centralized architecture. The localization is expanded in [75] and is
demonstrated on ground robots that also depend on high-contrast visual markers for range and bearing measurements and measurement correspondences. Our work extends the decentralized concept to UAS with only inter-vehicle range measurements and demonstrates how the optimization behaves with relatively low-bandwidth telemetry-radio communication. Although this work focuses on different aspects of the GPS-denied problem, we rely on their equivalency analysis to infer that a single vehicle’s solution is representative of the others (assuming they all utilize the same information). Further, this work utilizes only a subset of the measurement information to provide effective coordination but could utilize all the information they use if available.

This work was performed within the context of a multi-layer GPS-denied framework called relative navigation [26, 33]. It includes a real-time front end that navigates with respect to the local surroundings and a back end that is responsible for accomplishing the global mission. The front-end includes an EKF-based estimator that regularly resets its origin, by zeroing the aircraft position and heading, in coordination with visual keyframe declarations. Just prior to each reset step, the estimator publishes its transform and associated covariance to the back end. The back end represents the vehicle trajectory as a pose graph where each transformation is an edge in a graph and each front-end reset produces a new node, or global position variable, in the graph. The transforms are used in edge factors, where each edge relates two pose variables, or nodes, and calculates a loss. When the back-end is able to include other global constraints, such as loop closures and opportunistic global measurements, the graph becomes over constrained and can be optimized by minimizing the total loss of all the factors. Pose graph optimization has been studied extensively and several computationally efficient methods are available for performing the optimizations [41, 21].

Relative navigation has been tested extensively on multi-rotor aircraft [33] and recently on fixed-wing aircraft [64] and has several advantages for GPS-denied navigation. The front-end EKF is computationally tractable and operates in real-time. By allowing it to operate only with respect to a local keyframe, the estimator is observable and consistent by construction [30, 27]. The back-
Figure 6.1: The relative navigation architecture has been presented in prior work. It consists of a front end, that navigates relative to the the surroundings by regularly resetting the origin, and a back end, that accounts for the global state in a graph structure. This work introduces a slight modification to the architecture by adding a block for communication and ranging to other UAS. It adds range measurement and the other vehicles’ odometry to the back-end graph and requires the front-end estimator to coordinate resets with the other UAS.

end graph structure is able to more accurately represent the nonlinear uncertainty of the state due to errors in the yaw angle by concatenating edges published from the front-end estimator. The back-end also benefits from the presence of the front-end estimator because it effectively marginalizes a relatively large amount of sensor information into a single edge, making the back-end graph more sparse. Several seconds of IMU and image sensor data become a single delta-pose transformation with an associated covariance. This sparsity enables the back-end graph to scale to long trajectories and, in the context of this work, reduces the amount of information to be communicated between aircraft. This reduction will be discussed in more detail as part of the discussion of flight test results in Section 6.6.
Figure 6.2: Left: a timeline is depicted showing when nodes for two agents are declared. It also shows when a communication between agents takes place (red x). The communication is initiated by agent B and coincides with the declaration of node m. Right: the trajectory of the two agents (dashed lines) and the graph where each edge points to its associated node (arrows). When the communication takes place, another constraint is added to the graph in the form of an inter-vehicle measurement (red double arrow). In a step we call the coordinated reset, agent A must declare a node simultaneous to the communication for the inter-vehicle measurement to connect and constrain the graph.

6.3 Development

6.3.1 Architecture

The relative navigation architecture is shown in Figure 6.1. This work introduces a modification to the architecture to allow communication between vehicles. The communication block has two main purposes: 1) to share odometry edges, including change in position and associated uncertainties, from one vehicle’s front-end to another vehicle’s back-end and 2) to produce inter-vehicle measurement constraints. As a part of producing the inter-vehicle measurement, the communication block also performs a third minor function: when an inter-vehicle measurement is produced, it forces the front-end estimator to perform a reset step, declare a new node, and thus publish a new edge.

Whenever a measurement between vehicles is produced, both vehicles must also declare a new local origin. If no new origin is declared, the measurement factor in the graph will have no node to
which it can connect, and the vehicle trajectories will not be constrained. The timing of the vehicles reset step is shown in Figure 6.2. To reiterate, for two vehicles to incorporate an inter-vehicle range measurement they must also coordinate a reset in their respective front-end estimators. The front ends will then reset their global origins and publish their current state as an edge to be added to the graph. The current node will provide a suitable location for the inter-vehicle measurement to provide a constraint to the graph. We call this process the coordinated reset, because both aircraft perform a reset that is effectively simultaneous.

6.3.2 Inter-Vehicle Measurements and Messaging

This work assumes the aircraft have access to inter-vehicle range measurements. These measurements could be from specialized radios that produce distance measurements using time-of-flight, received-signal-strength-indicator (RSSI), or radio-frequency identification (RFID) [76] measurements. Measurement information could also be obtained from the carrier frequency of signals carrying additional information [77]. Range measurements to a fixed ground-based range station using a distance-measuring-equipment (DME) transponder have been used on aircraft for many years. Real-time location services (RTLS) represent a relatively new capability that is being applied to supply chains in warehouse environments [78]. Range measurements between small UAS, however, have been mainly demonstrated in laboratory experiments and remains a capability that is not readily available, though this capability has been assumed, and simulated, in various other works [13, 22, 73]. Range measurements in this work will be simulated by differencing the GPS-INS positions of the separate aircraft so that every communication exchange between them will produce a range measurement. This is as if the radio signal itself enabled the measurement by using its time of flight, for example, to measure the distance.

This work also limits the back-end graph to poses on a two dimensional plane. The prior work on front-end development computed a full 3D transformation for each edge, where each odometry edge consists of a four vector (consisting of $x$, $y$, $z$, and $\psi$) and a four-by-four covariance
matrix representing the uncertainty ellipse. In this work, we ignore the vertical \( z \) dimension of the published transformation and only utilize the horizontal \((x \text{ and } y)\) and yaw \((\psi)\) components of the transformation. This simplification is justified by the fact that the altitude of a UAS is observable with the use of barometric pressure altimeters and simple trigonometry can be used to remove the height component of any inter-vehicle measurement. Further, to minimize the amount of data necessary to communicate a given edge, the edge uncertainty was represented using only the diagonal elements of the covariance. This simplification is a slight mis-modeling of the uncertainty ellipse but is justified because, in general, the off-diagonal cross correlation terms remain small due to frequent resets in the front-end estimator. These simplifications are relatively common in various other solutions for GPS-denied navigation because they work well in practice, without oversimplifying the most challenging aspects of the GPS-denied navigation problem.

### 6.3.3 Communication Protocol

For vehicles to cooperatively navigate they need to share information using a defined communications protocol. The communication protocol is described below both for completeness and to emphasize the robustness of the system. While more complex and robust communication methods are common, the simple nature of the communication will highlight how the system performs when communication is limited or less than ideal due to dropouts and lost information.

To create a simple messaging system a number of conventions must first be defined. First, each vehicle is identified by a single 8-bit character or vehicle ID. The front-end estimator of each vehicle regularly publishes edges and each edge is identified by an index starting from the beginning of the flight trajectory. Each edge index also identifies an associated node in the graph, where the edge points to the node. Therefore, in a multi-vehicle scenario, a given edge is identified first by the vehicle from which it originated and then by the index in the list of edges that represent the vehicle trajectory. The edge itself consists of values for the change in north and east positions and yaw angle from the previous node to current node as well as three variances for the respective val-
ues. Finally, a range measurement is identified by the two vehicles from which it was produced and the index of the nodes (or edges) that it connects. For example, the range measurement depicted in Figure 6.2 would be identified with $A, n + 1, B, m$. The measurement consists of a distance and an assumed measurement variance. For convenience, edges also include the time when they were published from the front-end and ranges include the time of the measurement. Complete edge and range messages are defined in Tables 6.1 and 6.2 respectively.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle ID</td>
<td>char</td>
</tr>
<tr>
<td>Edge index</td>
<td>uint32</td>
</tr>
<tr>
<td>Delta pose (x, y, $\psi$)</td>
<td>float[3]</td>
</tr>
<tr>
<td>Variance ($\sigma_x$, $\sigma_y$, $\sigma_\psi$)</td>
<td>float[3]</td>
</tr>
<tr>
<td>Time stamp</td>
<td>uint64</td>
</tr>
</tbody>
</table>

**Table 6.2:** Range measurement message definition.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle 1 ID</td>
<td>char</td>
</tr>
<tr>
<td>Vehicle 1 edge index</td>
<td>uint32</td>
</tr>
<tr>
<td>Vehicle 2 ID</td>
<td>char</td>
</tr>
<tr>
<td>Vehicle 2 edge index</td>
<td>uint32</td>
</tr>
<tr>
<td>Distance</td>
<td>float</td>
</tr>
<tr>
<td>Time stamp</td>
<td>uint64</td>
</tr>
</tbody>
</table>

This work assumes that the various vehicles are not in constant communication. When a given aircraft is ready to communicate it sends a request to another vehicle, which may or may not respond. The request message first identifies the requesting and the target vehicle with vehicle IDs. Next the request includes the index of the last consecutive edge index for each vehicle in the swarm. The requesting vehicle does not request information about its own flight path because it will have the most current information from its own front-end estimator. The request also includes the earliest time stamp of the edges listed. This time stamp will allow the target vehicle to identify which range measurements to include in the response and will be discussed further below. Finally, since the range measurements are only simulated from the true positions of the aircraft, and we presume they would be created using the communication itself, the request also includes the true
north and east position and the edge index for which the measurement would be attached. The request message data types are contained in Table 6.3.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caller vehicle ID</td>
<td>char</td>
</tr>
<tr>
<td>Target vehicle ID</td>
<td>char</td>
</tr>
<tr>
<td>Vehicle ID list</td>
<td>char[n]</td>
</tr>
<tr>
<td>Last edge indices</td>
<td>uint32[n]</td>
</tr>
<tr>
<td>Time stamp of earliest edge</td>
<td>uint64</td>
</tr>
<tr>
<td>*True position (x, y)</td>
<td>float[2]</td>
</tr>
<tr>
<td>Edge index</td>
<td>uint32</td>
</tr>
</tbody>
</table>

$n$ is the number of vehicles in the swarm minus one.
* included to simulate a range measurement.

When responding to a request message, the responding aircraft uses its own true position to calculate the simulated range measurement that would, in reality, have been created from the communication signal itself. The response message also contains the rest of the range measurement bookkeeping information, including the caller and target IDs (which also serve to indicate the communicating aircraft), the measurement time stamp, and the edge index associated with the measurement for each aircraft. Finally, the response message includes the number of edges for each vehicle and total number of range measurements it will be sending in response. Immediately after the response message is sent, the list of edge and range measurements are returned. Note that the request and response messages are required for completing a range measurement but the following list of edges and previous range measurements are not. The response message data types are contained in Table 6.4.

6.4 Simulation

6.4.1 Description

A simulation was created for testing the proposed method. The simulation included several independent agents that navigated in two dimensions. Each agent used a simple wheeled robot
Table 6.4: Response message definition.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caller vehicle ID</td>
<td>char</td>
</tr>
<tr>
<td>Target vehicle ID</td>
<td>char</td>
</tr>
<tr>
<td>Caller vehicle edge index for range</td>
<td>uint32</td>
</tr>
<tr>
<td>Target vehicle edge index for range</td>
<td>uint32</td>
</tr>
<tr>
<td>*Distance</td>
<td>float</td>
</tr>
<tr>
<td>Range time stamp</td>
<td>uint64</td>
</tr>
<tr>
<td>Number of expected edge per vehicle</td>
<td>uint8[n]</td>
</tr>
<tr>
<td>Number of expected range messages</td>
<td>uint8</td>
</tr>
</tbody>
</table>

\(n\) is the number of vehicles in the swarm minus one.

* included to simulate a range measurement.

model from [4] and used the propagation and linearization step of an extended Kalman filter to simulate a noisy process.

The forward velocity and rotational noise parameters were adjusted so that when the front end performed a reset and declared a new node, the resulting covariance approximately matched that of the actual front-end estimator described in [64]. A motion profile was also created that simulated the motion profile of the actual aircraft, that is mostly straight with some side-to-side motion due to imprecise control and various disturbances. The simulation was given a perfect initialization and had no bias due to velocity error (both of which were seen in hardware testing of the filter in [64]).

The simulation also implemented separate back-end graphs for each vehicle. The simulation used GTSAM for graph-based optimization. The entire simulation is implemented in the MATLAB programming language and utilized the MATLAB wrappers for incorporating GTSAM functionality [41].

Each agent was implemented as a class object. The communication protocol described previously was implemented using function calls, where one agent could make a request by calling a member function of another agent. This means that communication in the simulation was perfect.
and instantaneous. The agents were also capable of coordinating resets with communication and range measurements.

The simulation was successfully tested with a small swarm of up to nine agents. In simulation, the proposed method scales well and could be extended to work on even more agents. The limiting factor in a real scenario would likely be the communication bandwidth because each agent must communicate the edges and measurements of all the other agents. The simulation results below utilize only three simulated agents for two reasons. First, an effort was made to have the simulation match the hardware results, which were limited to three agents, and second, because there is a diminishing return on improvement to localization by adding additional agents.

### 6.4.2 Results

The simulation was run with three agents cooperatively navigating. The results in Figure 6.3 show the agents navigating over a 2250 meter trajectory. The figure shows the actual, unoptimized, and cooperatively-optimized trajectories at the end of the flight. It also shows the error (optimized and unoptimized) of the center aircraft for the duration of the flight, because the graph is optimized
after every communication. This demonstrates that the graph is optimized incrementally throughout the flight, as opposed to only optimized at the end or in post process. The results shown are instructive because while the final optimized trajectory is much closer to the true trajectory, the optimized error is actually greater at around 60 seconds into the run. This is caused by errors in the other two aircraft temporarily pulling the center aircraft’s estimates away from the true position. These results demonstrate that global pose optimization is not guaranteed to produce a better localization result and we must therefore rely on a statistical analysis to show it will provide an improved localization result on average.

In general, the improvement for cooperative aircraft localization is expected to be approximately $\frac{1}{\sqrt{n}}$, where $n$ is the number of agents cooperating. This comes from assuming the navigation errors in the trajectories are purely random errors and that each aircraft trajectory is sampled from the same random process. In other words, the expectation of the standard deviation of the optimized result will be the expected standard deviation of the individual agents errors divided by $\sqrt{n}$. This also means in some cases the optimized result for a given aircraft may be worse although the error in the whole system is expected to be reduced.

The combined result of all the aircraft may actually have more improvement if the trajectories are not sampled from the same random process and there is an increase in observability in the system. For example, the increased observability may be due to the specific trajectories taken by the agents relative to each other. To utilize this improvement, however, the agents would be required to fly more varied paths. The observability would likely also improve by using the full covariance matrix for each of the graph edges.

The back end’s ability to remove error is correlated with the quality of the constraints that it creates between aircraft. For example, when the UAS are all flying in the same direction and at the same speed, inter-vehicle range measurements only help to remove error that is approximately perpendicular to the flight path. If instead, a single UAS flew perpendicular to the others, the
position accuracy would increase because the constraints on the graph would effectively provide more information as the observability of the system improves.

Monte-Carlo testing was used to quantify the expected benefit of the proposed system and application. Figure 6.4 shows the results of 3500 simulations of three agents. These results show a reduction in the total error on average by a factor of 0.64 compared to the theoretical expectation of \( \frac{1}{\sqrt{3}} \approx 0.58 \). The observability properties of the system are partially demonstrated by the fact that there is more error (and greater error reduction) in the \( x \) direction, which was perpendicular to the general motion of the agents.

The simulation results also show that the architecture is able to maintain the relative positions of the aircraft. Even when the optimization reduced the global accuracy of a given agent, due to drift errors in the other agent’s odometry or local minima in the optimization, the relative position of the agents is often well preserved. As discussed previously, depending on the mission scenario, the relative position of the agents may actually be more important than knowing the global position. Figure 6.5 shows that the optimization was able to, on average, decrease the relative error between agents. This plot shows the relative error in the agents reference frame at the end of the simulation.
The relative errors between the agents are shown, after Monte-Carlo simulation, in a polar plot (in degrees and meters). In a given agent’s reference frame, the error of the relative position of the other agents are plotted for pre-optimized (green) and optimized (magenta) estimates.

The relative position was produced by taking the vector from an agent to the another agent, in the first agent’s reference frame. The estimated vector was subtracted from the true vector to produce the points that are plotted in the chart for unoptimized and optimized estimates.

6.5 Hardware Experiments

6.5.1 Description

The front-end filter was implemented on the small, remotely-controlled, hobby-grade aircraft that is shown in Figure 6.6. The aircraft carried an NVIDIA Jetson TX2 embedded computer. The TX2 received images from a Point Grey Chameleon 3 USB camera and the acceleration and angular-rate gyroscope measurements from a thermally calibrated InertialSense IMU. This IMU is also a micro GPS-INS and is capable of producing a full navigation solution for truth comparison. The implementation details of the front-end filter are described in [64].

Due to the complexity and regulatory barriers of flying multiple autonomous aircraft simultaneously, we elected to use a single aircraft with a complete front-end estimator and back-end
Figure 6.6: The method was tested on a modified STRIX StratoSurfer, a 1.5 m wingspan aircraft. The single aircraft was flown over a 6 km trajectory, in multiple test flights and the front-end odometry was recorded. To facilitate a multi-vehicle scenario, the architecture was implemented on a UAS that was also in communication with a ground based computer that was replaying the previously recorded data as a surrogate for additional aircraft.
map. In addition to this aircraft, we created the other aircraft virtually using recorded front-end data, including edge transformations and true states. The recorded data was produced by flying the test aircraft over the flight trajectory over several previous flights. In the multi-vehicle test, the front-end estimates from the recorded flights were replayed on a ground computer that was able to communicate with the actual flying aircraft. From the test aircraft’s perspective, other than the simulation of range measurements by differencing the true positions, the virtual aircraft accurately mimicked real aircraft, each having a front-end estimator, back-end graph, and communication node. Because actual communication to and from the ground computer was used, the use of recorded agents did not detract from the experiment’s ability to show the advantage of the system even with limited or intermittent communication.

The front-end estimator was modified to enable the coordinated reset. The test aircraft’s front end performed a coordinated reset whenever a recorded agent initiated a communication request with it. The coordinated resets, however, are not possible for agents with prerecorded front-end data because the reset steps are in reality performed prior to the multi-vehicle flight test. The recorded agents instead applied the simulated range measurements to the most recent node. For example, in Figure 6.2, if agent A was created from recorded odometry then the measurement would be applied to edge n and the true position of node n would be used to simulate the measurement. This is possible because of the division between the relative front end and the global back end. The division allows the back-end graph to be optimized without any concept of time (i.e. there is no concept of simultaneous events in a pose graph). In other words, timing can be stretched in the graph without consequence to the real-time operation of the vehicle.

The communication protocol was implemented using MAVLink, an open-source messaging library that is part of the Drone Code project and managed by the Linux Foundation. The messages were sent using 915 MHz SiK radios. The radios are inexpensive and widely used for telemetry communication on hobby-grade aircraft.
The resulting graph computed on board the test aircraft at the end of the 6-km flight. The unoptimized odometry included several sources of error, including initialization error and scale bias [64]. The optimization of the graph is able to remove the initialization error and maintain the relative position of the aircraft, although a scale error is still apparent.

The front-end estimator and back-end graph were implemented in the C++ programming language which provided both computational efficiency and speed. The communication protocol, messaging, and range-measurement simulation were implemented in Python. Robot Operating System (ROS) was used for programming language interoperability and launching the various pieces of the architecture for each agent. ROS functionality was also used for recording and playing back the front-end estimate for the recorded agents.

### 6.5.2 Results

The complete hardware results are shown in Figure 6.7. The results shown are the from back-end graph that was produced on the center aircraft while in flight. The unoptimized front-end odometry for the two side aircraft was from recorded data that was played back on the ground-based computer.

In these results the center aircraft had a large initial heading error and the top/right aircraft had a bias in its odometry that resulted from an initialization error. All three aircraft also have a scale error due to a forward velocity bias. These types of errors, including their causes and
possible solutions, come from the front-end estimator and are discussed in detail in [64]. Despite
the presence of these errors in the front-end odometry, the cooperative back end is able to optimize
and produce an improved solution for the aircraft odometry. As in the simulation, the relative
positions of the aircraft are also preserved in the optimized graph.

As the center aircraft was flown, it had several communication dropouts where it was unable
to receive the requested information from the other aircraft. Despite this taking place, the system
was able to exchange and incorporate the missing information when communication was fully
restored and optimize it appropriately. Figure 6.8 shows an example of a dropout event that lasted
for approximately 40 seconds. In the dropout period several partial communications were made
but were unable to be incorporated into the graph because of missing data. Figure 6.8 shows the
center aircraft’s back-end graph just before and directly after a complete communication update
where all missing information is back filled and the graph is subsequently optimized.

Figure 6.8: The test aircraft (center flight path) experienced significant communication dropouts during the flight. The left plot shows just prior to a communication update and the right plot shows approximately a second later, just after a successful communication was achieved. The update back-fills the graph with approximately 40 seconds of odometry data from the other two UAS and it enables the usage of ten previous range measurements between them. The graph is also updated and optimized to improve, although only slightly in this case, the global position of the UAS.
Figure 6.9: A cooperative graph was constructed using the proposed system and three previously recorded flights. Left: the graph includes trajectories with significant scale bias and horizontal drift. Right: allowing the center aircraft to include two simulated GPS measurements in the graph improves the localization of all three aircraft significantly.

6.6 Post-Processed Experiments

There are several aspects of this work that are worth exploring beyond what the simulation and hardware results display. First, the graph is capable of including other opportunistic measurements beyond inter-vehicle range measurements alone. Several examples of additional measurements and their benefits are briefly explored in [64]. Figure 6.9 provides an example of a scenario where the center aircraft receives intermittent, simulated global position measurements such as those from GPS or a computationally expensive satellite-image-based place recognition system [9].

The results in Figure 6.9 were produced by using three previously flown aircraft trajectories and combining them using the proposed method, including full communication between agents, in a post-process computation. Each trajectory includes significant initialization errors and bias scale. The results on the left show the optimized graph without GPS measurements and provide an example of the optimization converging on a local minima, although the final position of the agents relative to each other is well maintained. The results on the right show that all three aircraft receive the benefit of just two global measurements by removing both scale error due to bias and lateral error due to drift.
Also, it is worth noting the amount of data that is transmitted between agents for creating the complete, multi-agent, back-end graph. Table 6.5, below, shows the amount of data that was sent and received by the center aircraft during the full flight tests. The ideal amount of communication was calculated while the aircraft was on the ground and using wired connections (and recorded data) instead of the SiK radios. The total real communication is larger due to dropped messages and having to resend data when it is not successfully received.

<table>
<thead>
<tr>
<th></th>
<th>Sent</th>
<th>Received</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal</td>
<td>31.3 KB</td>
<td>19.4 KB</td>
</tr>
<tr>
<td>Real</td>
<td>54.0 KB</td>
<td>37.6 KB</td>
</tr>
</tbody>
</table>

Table 6.5: Amount of communicated data during test.

A rough estimate of the total amount of sensor data processed, including imagery and IMU measurements, is 5.8 GB for all three vehicles, whereas the back-end graph is constructed from less than 0.15 MB of data. A single aircraft actually sends less than 0.15 MB because of the simplifications made in the communications protocol and because they do not have to request data about their own trajectory. The reduction from total measurement data to the data communicated highlights the efficiency of this approach and its potential for scalability to more agents without significant limitations on communications.

6.7 Conclusion

UAS can benefit from cooperative navigation. In GPS-denied environments, where sensor noise and bias can cause an agent’s estimates to drift, sharing information between aircraft can enable an improvement in localization accuracy.

This paper has demonstrated a method for fixed-wing UAS to benefit from cooperative localization. This was accomplished by utilizing the relative navigation framework and a previously proposed front-end estimator for vehicle odometry estimation. Each aircraft was able to perform decentralized, pose-graph optimization by utilizing a modification to the relative navigation archi-
A multi-agent simulation was able to demonstrate the value of the proposed method through Monte-Carlo testing. It demonstrated that cooperation does reduce the expected value of the total localization error of the swarm although an individual aircraft’s localization is not guaranteed to improve. Further the simulation showed that cooperation helps maintain the formation, or the positions of the aircraft relative to each other, which is often essential for achieving a multi-vehicle mission objective.

The full hardware flight-test demonstration further showed the benefits of the proposed method. The demonstration utilized a flight vehicle, complete with front-end estimator and back-end graph, communicating with agents that were simulated from previously recorded front-end data. The back-end graph optimization was able to remove gross initialization and bias errors that were not present in the simulation tests. Because the flight tests included limited communications capabilities, including the use of hobby-grade, telemetry radios, the tests were able to demonstrate that the method performed well despite extended periods of communications dropout. During periods of communication dropout, the method allowed the effected aircraft to revert to relying on its own front-end odometry and then, once communication was restored, it regained the advantages of cooperation.

The value of the proposed method is further shown by additional experiments that were performed by post-processing the flight data. First, these experiments showed that the purposed method greatly reduces the amount of data communicated between aircraft. This means there is potential for extending it to more aircraft, leading to better results. Further, the experiments showed that if a single aircraft can utilize global measurements, then the localization of the other aircraft is significantly improved as well.
Acknowledgment

This work has been funded by the Center for Unmanned Aircraft Systems (C-UAS), a National Science Foundation Industry University Cooperative Research Center (IUCRC) under NSF award No. IIP-1650547 along with significant contributions from the AFRL Munitions Directorate and other C-UAS industry members. Gary Ellingson also received support from the Utah NASA Space Grant Consortium Fellowship.
CHAPTER 7. CONCLUSION AND FUTURE WORK

The goal of this dissertation is to contribute to the maturation of small unmanned aircraft. Before introduction into the national airspace or use in military applications, small unmanned aircraft will need greater reliability and to be robust to GPS signal degradation and dropout. This research has expanded the capabilities of these aircraft in several ways. First, it has experimentally validated the relative navigation architecture, which has many advantages for GPS-denied/degraded operations. Relative navigation was then extended to fixed-wing aircraft through the development of a tightly coupled visual-inertial odometry. The odometry accounts for fixed-wing sensing requirements and flight profiles and uses only camera and IMU without assumptions about the distance to observed features. Finally, the relative navigation architecture was modified to enable multiple aircraft to cooperatively navigate. The proposed method is decentralized and robust to communications dropouts and delays. The following sections first discuss the relevant assumptions and limitations for which the work is applicable and then outline several additional ways that the work completed in this dissertation could be used and extended.

7.1 Assumptions and Limitations

This work has focused on extending relative navigation to fixed-wing aircraft as well as multiple aircraft navigating cooperatively. While the presented VIO method has been shown to be adequate based on the sensing requirements of fixed-wing MAVs, it is not strictly limited to this type of vehicle. It would likely work on rockets, helicopters, blimps, and even multi-rotor MAVs as long as several requirements are satisfied. For the VIO to produce good results, it needs to have adequate excitation from the vehicle motion, a requirement that is similar to the observability
issues associated with straight-and-level flight that have been discussed in detail in the proceeding chapters.

The VIO also requires there to be image features moving past the camera. Though it may seem obvious, the filter will fail and the estimates diverge if the feature tracker is incapable of tracking features due to clouds, smoke, or a featureless background such as glassy water. Another failure mode is illustrated by a scenario with the vehicle continually hovering in place. Hypothetically, this would cause the feature tracks to never terminate, and therefore provide no updates to the filter. Similarly, if the vehicle was at an extremely high altitude and traveling slowly, the apparent motion of the features would be rotation only, which would make the VIO incapable of resolving scale.

There are also several practical limitations to the proposed multi-vehicle method. First the amount of delay caused by dropped communication and the relatively computationally intense graph optimization that can be tolerated is limited for a given mission objective. For example, if the objective requires the cooperative MAVs have precise navigation, with errors on the order of decimeters, then a 45 second communications delay would be unacceptable. The rate of inter-vehicle communication is another potential limitation. For example, if the vehicles attempt to communicate with every image, making the coordinated reset declare each image as a keyframe, then the front-end estimator will not be able to provide good estimates and the back-end graph will include too many factors for efficient optimization.

7.2 Planning and Control

This work has focused on navigation and localization of fixed-wing MAVs but the experiments were flown manually and did not include the planner and controller necessary to fly autonomously. Thus far the division of back-end global planner and the front-end local controller, that would be necessary for completing a fully autonomous mission, has not been explored for a fixed-wing UAS.
The controller for cooperative navigation has also not been explored. The vehicles need a way to cooperatively complete a mission as well as de-conflict trajectories that would cause a crash. A centralized planner would be straightforward but would go against the decentralized and opportunistic nature of the proposed method. More elegant would be a decentralized planner that uses a consensus algorithm to help all the vehicles arrive at the same plan. This would require a major modification to the proposed communication scheme and would require significant testing to ensure the solution remains robust to communications dropout and delay.

The cooperative controller could also be constructed to improve accuracy in the back-end graph. The graph's ability to remove accumulated drift is correlated with the quality of the constraints that it creates between aircraft. For example, when the UAS are all flying in the same direction and at the same speed, inter-vehicle range measurement only help to remove yaw error. If instead a single UAS flew perpendicular to the others, the position accuracy would also increase because the constraints on the graph would effectively provide more information. A controller could be developed to exploit this phenomenon. It could, for example, periodically have one agent turn sharply or even vary the commanded airspeed between agents to increase the accuracy of estimation.

7.3 Front-End Estimator

The next group of potential improvements could focus on trying other front-end odometry methods. The choice of the MSCKF measurement model has many advantages, especially for relative navigation, but other visual-inertial odometry methods may be more accurate at the price of more computation [45].

The forward velocity unobservability that is experienced in straight-and-level flight can be remedied by controlling the UAS such that it remains observable, as we have demonstrated, or perhaps by adding additional sensing modalities. Feasible options include airspeed sensors or single-point depth sensors. An airspeed sensor would measure the forward velocity with respect
to the airmass and therefore would require modeling the wind. Relying on a depth sensor, such as a sonar or single laser altimeter, could improve the scale of the estimates by adding constraints on the distance to observed features. A model for combining a depth sensor in combination with a camera is presented in [79]. This type of approach could be applied to the MSCKF measurements. The MSCKF performs a least-squares optimization to solve for the feature location. A distance constraint provided by the sensor in the optimization could improve estimates of the scale and velocity of the aircraft.

7.4 Map Generation

The back-end map could also be improved to better utilize the information gathered. The current capability primarily uses the map for characterizing the localization error. The visual map could be created to facilitate completion of various mission objectives, including surveillance, target identification, and tracking. A simple but effective way to accomplish this would be by tiling the keyframe images to create a visual representation of the terrain. The homographies used to tile the keyframe images could also be used as constraints in the back-end optimization to improve global estimates.

Tiling the images for map creation requires the knowledge of distance to the images as well as estimates from one image to the next. Simple approaches use the altitude of the aircraft to obtain the distance and then assume the terrain is flat, using the so-called flat-earth model. In our approach the front end provides the inter-keyframe transforms. The MSCKF measurement model also solves for the location of the observed features in the keyframe coordinate frame. This means the image provides dense appearance information and the features become a sparse point cloud with corresponding location information. If the terrain is fairly smooth, sharp cliffs or jagged corners, an image could be draped over the point cloud to provide a rough shape of the terrain for a given keyframe image. The separate keyframe images could then be combined using a combination of iterative closest point and image homography methods. Map creation, however,
greatly increases the complexity of sharing information between cooperative agents and would require changes to the communication approach.

7.5 Back-End Measurements

Exploring various other types of back-end measurements, as well as alternate use cases, may highlight the practical advantages of the proposed method. A potential scenario was partially demonstrated in a Chapter 6, where only one vehicle had access to global measurements and the rest of the cooperating vehicles gained the localization advantages of those measurements. A small alteration to the scenario would be to have one vehicle use a global measurement from a satellite image-based localization. A similar method is used in [9] to initialize a filter of a spacecraft approaching a planetary body. Their approach uses prerecorded satellite images to find the current location of the aircraft and is computationally expensive.

Another relevant scenario would be applying the relative navigation architecture and cooperative localizing when multiple vehicles are trying to achieve a goal location, are performing target tracking, or are prosecuting an enemy vehicle. The scenario could require that only some of the vehicles have the ability to measure the location of the target or objective. The blind vehicles would be required to rely on the targeting capabilities of one or more other UAS to know their relative position. Since the inter-vehicle relative position is maintained by the proposed method then this may be an effective way to increase the capability of the less capable agents.
REFERENCES


