2015-05-01

# Globally Consistent Map Generation in GPS-Degraded Environments

Paul William Nyholm
*Brigham Young University - Provo*

# Globally Consistent Map Generation in GPS-Degraded Environments

Paul William Nyholm

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Timothy W. McLain, Chair
Randal W. Beard
Mark B. Colton

Department of Mechanical Engineering

Brigham Young University

May 2015

ABSTRACT

Globally Consistent Map Generation in GPS-Degraded Environments

Paul William Nyholm
Department of Mechanical Engineering, BYU
Master of Science

Heavy reliance on GPS is preventing unmanned air systems (UAS) from being fully integrated for many of their numerous applications. In the absence of GPS, GPS-reliant UAS have difficulty estimating vehicle states resulting in vehicle failures. Additionally, naively using erroneous measurements when GPS is available can result in significant state inaccuracies. We present a simultaneous localization and mapping (SLAM) solution to GPS-degraded navigation that allows vehicle state estimation and control independent of global information. Optionally, a global map can be constructed from odometry measurements and can be updated with GPS measurements while maintaining robustness against outliers.

We detail a relative navigation SLAM framework that distinguishes a relative front end and global back end. It decouples the front-end flight critical processes, such as state estimation and control, from back-end global map construction and optimization. Components of the front end function relative to a locally-established coordinate frame, completely independent from global state information. The approach maintains state estimation continuity in the absence of GPS measurements or when there are jumps in the global state, such as after map optimization. A global graph-based SLAM back end complements the relative front end by constructing and refining a global map using odometry measurements provided by the front end.

Unlike typical approaches that use GPS in the front end to estimate global states, our unique back end uses a virtual zero and virtual constraint to allow intermittent GPS measurements to be applied directly to the map. Methods are presented to reduce the scale of GPS induced costs and refine the map's initial orientation prior to optimization, both of which facilitate convergence to a globally consistent map. The approach uses a state-of-the-art robust least-squares optimization algorithm called dynamic covariance scaling (DCS) to identify and reject outlying GPS measurements and loop closures. We demonstrate our system's ability to generate globally consistent and aligned maps in GPS-degraded environments through simulation, hand-carried, and flight test results.

Keywords: simultaneous localization and mapping, SLAM, indoor flight, outdoor flight, GPS-denied flight, autonomous flight, navigation, intermittent GPS, degraded GPS, back-end optimization

# ACKNOWLEDGMENTS

I would like to take this opportunity to thank all those who have helped and supported me throughout this research. My advisor Dr. McLain, and committee members Dr. Beard and Dr. Colton have taught me much through coursework as well as provided research guidance and direction throughout the course of my studies. I am also grateful for the Center for Unmanned Aircraft Systems (C-UAS) and its participants for funding this project and providing feedback on the work.

Without the help of the rest of my research team, David Wheeler, Dan Koch, Gary Ellingson, and James Jackson, this thesis would not have been possible. Likewise, safety pilots Dallin Briggs and Brandon Reimschissel were instrumental throughout our experimental flight tests. I am also thankful to all the MAGICCians that have supported and aided me during the past few years in the MAGICC Lab, in particularly Justin Mackay, Robert Klaus, and Peter Niedfeldt.

Lastly, I would like to acknowledge and thank my loving wife Danielle for her constant support and sacrifice to make this research possible; my son Brady for giving me a joyful, healthy distraction from research; and my parents Robert and Janet, who have always supported and encouraged my education.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# CHAPTER 1. INTRODUCTION

## 1.1 Motivation

Unmanned aircraft are becoming increasingly popular for a wide range of military and domestic applications, including search and rescue, agriculture and infrastructure monitoring, cinematography, and even product delivery [1,2]. However, for unmanned aircraft systems (UAS) to be widely used in such applications, both technological and legislative hurdles need to be overcome. A major obstacle hindering legislative approval in the United States is the issue of maintaining standards of safety when integrating UAS in the national airspace. Of particular importance is the vehicle's ability to operate safely in unknown and varied environments. Many systems are heavily reliant on external methods of localization such as motion capture systems or the Global Positioning System (GPS). Other approaches make strong assumptions about the particular environment they are intended to operate in. Vehicles attempting to operate outside of their assumed environment, or without expected methods of localization, are unable to perform as intended, potentially resulting in catastrophic crashes. Although systems are capable of autonomous navigation under specific circumstances, the development of UAS that are robust to unknown and varying environments is an open and active field of research.

One notable issue preventing the robustness necessary to integrate unmanned aircraft in the national airspace is heavy reliance on GPS. The United States joint chief of staff recently stated, "It seems critical to me that the Joint force should reduce its dependence on GPS-aided precision navigation and timing, allowing it to ultimately become less vulnerable, yet equally precise, and more resilient" [3]. In addition to heading and ground speed estimates, GPS provides low-rate position measurements that constrain drift in state estimates cause by noisy sensors like inertial measurement units (IMUs). Because of various sources of errors in GPS measurements, such as those listed in Table 1.1, robust autonomous systems cannot rely on GPS accuracy. Although the vehicle may be well informed about common GPS errors, such as atmospheric delays and dilution

of precision, environment specific errors, like multipath, shadowing, and jamming, are much more difficult to detect and mitigate. Even a brief loss or degradation of GPS is known to cause vehicle failures, often resulting in a crash. Consequently, GPS-denied navigation has been a focus of autonomous navigation research over the last decade.

Simultaneous localization and mapping (SLAM) addresses the GPS denied problem. SLAM solutions seek to build a map of a vehicle's trajectory throughout its flight while simultaneously localizing itself within the map, often independent of a priori information or GPS. This research seeks to validate a novel relative navigation SLAM solution, introduced by MAGICC Lab alumnus Robert Leishman [4], through implementation and experimentation. Additonally, we present and demonstrate a unique SLAM back end to integrate intermittent and degraded GPS in our navigation scheme without relying on it for flight critical processes, such as state estimation and control.

In the remainder of this chapter we discuss graph-based SLAM and why it is well suited for our navigation solution. We conclude with a summary of contributions presented in this thesis. Chapter 2 describes the development tools utilized throughout this research, including the multirotor aircraft used to perform flight tests. In Chapter 3 we present the relative navigation framework with results from indoor, GPS-denied flight tests. Chapter 4 details our unique approach to incorporating intermittent and degraded GPS measurements in the back end and presents validating simulation and experimental results. We conclude and present avenues for future work in Chapter 5.

Table 1.1: Sources of GPS Uncertainty

| Type | Description |
|---|---|
| Multi-path | Signal bounces before reaching receiver (false psuedo-range). |
| Number of satellites | Few visible satellites increase sensitivity to timing errors. |
| Dilution of precision | Visible satellites are poorly spaced. |
| Spoofing | Signal is locally recreated with false information. |
| Atmospheric delays | Signal is delayed due to ionsphere and troposhere influences. |

## 1.2 Background

The goal of autonomous navigation is for a vehicle to safely reach a goal location with limited human interaction. Such a vehicle must be capable of computing a path from a starting location to a goal location while avoiding obstacles and potential collisions. The literature refers to this as path planning. To plan paths, a map of the environment is required. Without GPS or a priori map knowledge, the typical approach is to have the vehicle accumulate environment information throughout its mission to build a map. The vehicle needs to be localized, or have a good sense of where it is within the map, to compute and traverse a path within it. When both map making and localization are being executed simultaneously, it is termed SLAM.

Autonomous navigation systems for ground vehicles have been successfully developed; however, interest in aerial vehicles exists because they provide significant advantages over their ground-based counterparts. Particularly in cluttered environments, aerial vehicles benefit by being unconstrained from the ground, allowing them to fly over difficult obstacles rather than navigating through them (e.g., staircase, debris).

Certain challenges arise when migrating autonomous ground vehicle technology to aerial vehicles because of the limited payload constraints, complex dynamics, and increased degrees of freedom of unmanned air vehicles (UAV). Ground-based robots are constrained to the ground and can therefore use encoder measurements from wheels to produce accurate odometry measurements. Conversely, because of the unconstrained 3D nature of UAVs, they must rely completely on other sensors, such as laser scanners, cameras, and IMUs. UAVs have inherent size, weight and power (SWAP) constraints forcing autonomous UAVs to utilize lightweight sensors. Technology for these lightweight sensors is not as accurate as the heavier, state-of-the-art alternatives. For example, SICK laser scanners are common on ground robots because of their high precision and wide range, but can weigh up to 7.5 kg [5] —clearly unsuitable for UAVs, which typically have payloads of less than 1 kg.

In GPS-denied environments, small UAS lack a direct global position measurement to constrain position estimates. Consequently, they rely on lightweight exteroceptive sensors (e.g., laser scanners, cameras) supplemented by interoceptive sensors (e.g., IMU) to estimate states. Exteroceptive sensors provide relative measurements that can be compounded, or summed, to estimate vehicle states; however, errors in relative measurements accumulate throughout a UAV's flight,

causing drift while performing SLAM. The drift is particularly evident when a UAS has returned to a location in the environment that it has already visited, known as loop closure or closing the loop.

Since the inception of SLAM in 1986, three main approaches to solving the SLAM problem have been developed [6]. The first approach in the literature uses an extended Kalman filter (EKF) to estimate the global position of landmarks or features and is appropriately coined EKF-SLAM. FastSLAM, a paradigm using the Rao-Blackwellized particle filter, then emerged, proving advantageous to EKF-SLAM in terms of computational demand but presenting problems of its own because of necessary resampling. MacDonald points out that both EKF-SLAM and FastSLAM are feature based, meaning that they reduce raw sensor measurements into a set of features; this renders the map unsuitable for high level path planning or other high level functions [7]. An alternative to these feature based methods was presented in 1997 by Lu and Milios who introduced a SLAM model using graph theory [8]. Recent advances in solving sparse systems have made graph-based SLAM an appealing technique that is "state-of-the-art with respect to speed and accuracy" [9].

### 1.2.1 Graph-based SLAM

Graph-based SLAM consists of a front end for graph construction and a back end for graph refinement and optimization. The SLAM front end is primarily responsible for fusing sensor information and estimating the vehicle's pose. Variations of the Kalman filter are heavily utilized in the literature for vehicle pose estimation. The back end stores estimated global vehicle poses as nodes in a graph. Graph edges encode both measurements and measurement uncertainty. Relative transformations between two consecutive poses can be computed and are added to the graph as odometry edges. A pose graph can be used to represent both 2D and 3D maps.

Additional edges can be established between nonconsecutive poses through loop closures. Loop closures occur when a vehicle returns to the same physical location more than once and are detected using exteroceptive sensors and place recognition algorithms, such as FAB-MAP [10] and BRIEF-Gist [11]. A loop closure edge encodes the relative transformation between two nonconsecutive poses that lie in the same physical location. See Section 3.3.1 for more details on graph construction.

Due to sensor noise, front-end pose estimates will drift from truth in the absence of GPS, as seen in Figure 1.1. A least-squares optimization problem can be formulated from a pose graph to compensate for drift and maintain a globally consistent map. The least-squares problem seeks to solve the most likely arrangement of poses given a set of measurements (odometry and loop closure edges). Graph nodes and edges become free variables and optimization constraints respectively. Many efficient graph optimization algorithms have been developed, several of which are open source and available online [12].

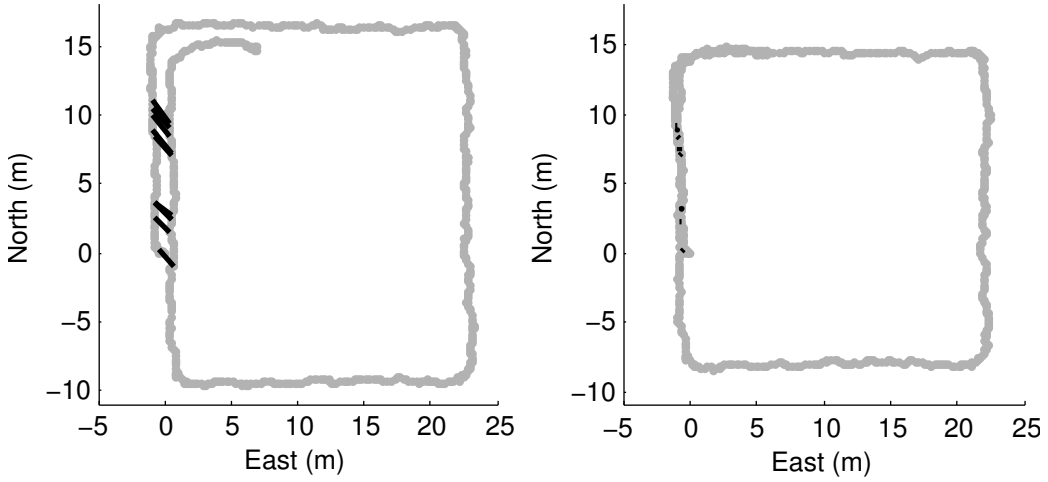



Figure 1.1: Map constructed from front-end odometry measurements, where nodes are spaced approximately 0.25 meters apart, before (left) and after (right) least-squares optimization. A multirotor aircraft was flown in a rectangular path through building hallways. Black lines indicate loop closures between the two connected nodes. The drift induced by odometry errors is significantly mitigated in the optimized graph.

There have been impressive autonomous UAS solutions for GPS-denied navigation that leverage various graph-based SLAM implementations. Grzonka et al. [13] use a laser range scanner and scan matching algorithm to estimate vehicle poses. For map optimization, they use the TORO [14] algorithm and are able to produce accurate maps of indoor environments. Similarly, Bachrach et al. [15] use a laser scanner based front end and use iSAM [16] for graph-based back end optimization. They show globally consistent maps created from experiments in both indoor and urban canyon environments. Fraundorfer et al. [17] show the feasibility of a UAS featuring a forward looking stereo camera as its primary exteroceptive sensor and use g2o [18] to optimize a

pose graph. These works show the feasibility of creating globally consistent maps, both indoors and outdoors, but they do not take advantage of GPS even when it is available.

Furthermore, least-squares optimization is inherently sensitive to outliers. A single false-positive loop closure can catastrophically derail the optimization solution. As such, most place recognition algorithms are tuned to return a match only when they are highly confident, but there is no theoretical guarantee that false-positives will not be returned. Our work focuses on leveraging GPS measurements when they are available, but not relying on them for state estimation or control. Additionally, we have formulated a back end that is robust to outlying measurements, specifically loop closures and GPS, that can be run real-time, onboard a UAS.

In our solution, we use a relative multiplicative extended Kalman filter to estimate states relative to locally established coordinate frames. In this way it is straightforward to output the transformation between coordinate frames as odometry measurements for 2D pose-graph construction and optimization. Using this approach, our state estimates and control can be completely decoupled from global measurement updates that are prone to gross errors, such as GPS. We take advantage of GPS measurements by applying them as direct global measurements in the back end, enabling accurate reconstruction of a vehicle's trajectory. We propose methods to improve optimization performance: first, to ensure that GPS induced errors are suitably scaled for optimization. And second, to refine the initial orientation of the graph to avoid local minima. Further, we implement dynamic covariance scaling to detect and reject outlying measurements. This relative navigation framework is validated through both indoor and outdoor experimental flight tests.

### 1.3 Contributions

Primary contributions resulting from this thesis include:

- Validation of the relative navigation framework for both indoor and outdoor flights. In particular, real-time, onboard place recognition and map optimization resulting in a globally consistent map.
- A unique back end capable of leveraging GPS measurements while detecting and rejecting outlying measurements.

- A back-end method to ensure that GPS errors are tractable and result in positive-semi-definite Hessian matrix formation.
- An algorithm to avoid local minima during optimization by adjusting a graph's initial orientation prior to optimization.

The following are tools that were developed throughout this research and can continue to be of use in related areas. All of these tools can be accessed through the MAGICC Lab server, http://magiccvs.byu.edu. A modular software architecture implementation of the relative navigation framework is available that allows plug-and-play capabilities of estimators, controllers, path planners, and sensors. A Robot Operating System (ROS) based aircraft simulator was developed for software-in-the-loop simulation that allows rapid prototyping and testing of controllers and path planners. And lastly, a 2D back end, implemented in both MATLAB and C++, that can optimize both simulated and real vehicle trajectories and measurements.

# CHAPTER 2. DEVELOPMENT TOOLS

This chapter begins by providing details about the multirotor aircraft used for flight experiments in this work. We then describe the motion capture system used for preliminary flight results. A brief overview of ROS is given followed by a description of the software-in-the-loop multirotor simulator. Finally, the MATLAB back end simulator is presented.

## 2.1 Hardware

The modified MikroKopter HexaKopter LX chassis shown in Figure 2.1 is the basis of our experimental flight platform. While some of the preliminary flight experiments in this thesis were performed using a MikroKopter FlightCtrl 2.1, with accompanying BL-Ctrl electronic speed controllers, we have since migrated to 3D Robotics' Pixhawk autopilot. We opt to use the Pixhawk because it and its code base have been released as an open-source project. This allows substantial flexibility while implementing custom attitude estimators and controllers. Afro SimonK ESCs are used in conjunction with the Pixhawk autopilot to control the six MikroKopter MK3638 brushless motors wielding 12 inch propellers.

As this is a development platform, we carry an Intel i7 processor on board, running Ubuntu 14.04 and ROS, to perform computationally expensive operations such as visual odometry, as well as to allow rapid development and experimentation of state estimators, controllers, and other modules. As our state estimation and control schemes mature we plan to migrate our code base to the Pixhawk autopilot, reducing the number of communication interfaces between state estimates and low-level attitude control.

We carry an array of sensors used for state estimation. A MEMs based IMU MicroStrain 3DM-GX3-15 provides the state estimator with accelerometer and gyroscope measurements at 100 Hz.

Figure 2.1: Multirotor aircraft used for flight experiments and demonstrations. The platform is equipped with the hardware listed in Table 2.1

The LV-MaxSonar-EZ3 ultrasonic sensor is mounted beneath the multirotor chassis to serve as an altimeter. It communicates directly with the Pixhawk (20 Hz), which passes messages to the relative state estimator to provide altitude measurements relative to the ground. We use covariance gating to reject outlying measurements.

An Asus Xtion Pro Live RGB-D camera is used to provide both RGB and depth images to a visual odometry algorithm at 15 to 30 Hz. The Xtion is a light-weight, low-power, RGB-D sensor that performs quite well indoors but suffers drastically as we move outdoors. Because the depth images are built from infrared structured light, infrared scatter from the sun bombards the sensor rendering it useless outdoors during daylight hours. Additionally, the limited range of the depth sensor, only 3.5 meters, it is not an ideal sensor for relatively uncluttered environments.

We are highly anticipating the arrival of a preordered Velodyne VLP-16 (puck), a small, light-weight 3D laser range finder suitable for small UAS applications. The laser scanner boasts 100+ meter accuracy, with 360 degree horizontal field of view, and 30 degree vertical field of view. Outdoor flight capabilities are expected to be greatly enhanced when this sensor is integrated with the current system.

A pseudo-consumer grade GPS module, the Ublox LEA-6T, is used to collect GPS measurements when available at up to 5 Hz. Although this unit is capable of outputting raw pseudorange measurements, we use the GPS navigation solution exclusively in our tests to show that our method of GPS integration can work with any consumer-grade module.

Table 2.1 briefly details the mentioned hardware. We estimate that the hexacopter is capable of carrying a 1 kilogram payload in addition to its current sensor suite. Two 4S 6000 mAh lithium polymer batteries are used in parallel to achieve vehicle flight times of approximately 12 minutes.

Table 2.1: Hardware Details

| Component | Description |
|---|---|
| Vehicle | Mikrokopter HexaKopter XL |
| Autopilot | 3DR Pixhawk |
| ESC | Afro SimonK |
| Motors | MikroKopter MK3833 |
| RGB-D Camera | ASUS Xtion Pro Live |
| IMU | MicroStrain 3DM-GX3-15 |
| Ultrasonic Altimeter | LV-MaxSonar-EZ3 |
| GPS Module | Ublox LEA-6T |
| Processor | Intel Core i7-2710QE (2.1GHz $\times$ 4) |
| LIDAR (expected) | Velodyne VLP-16 |

## 2.2 Motion Capture System

We use a motion capture system from Motion Analysis for accurate, potentially submillimeter, state estimates. This ground truth data is very useful for developing and testing both estimators and controllers. We can compare our front end state estimates to the motion capture truth to validate their accuracy. Controllers can be tested, decoupled from state filters, by supplying the camera's truth information for state control.

The system consists of an array of eight Motion Analysis HWK-200RT HAWK cameras, along with the Cortex software. A model of the vehicle to be tracked is created in Cortex by arranging a set of infrared reflector dots, or marker set, on the vehicle. The system provides position estimates of the marker set, and obtains velocity and acceleration information by differentiating the position. These states are broadcast on the local area network. A third-party software, Evart Bridge, is used to relay state information from the network to ROS, and publishes the data in a ROS friendly format at approximately 100 Hz.

## 2.3 ROS

Robot Operating System (ROS) [19] is an open-source development environment aimed at facilitating rapid prototyping and development of robotics. ROS is developed and supported primarily on Ubuntu. It facilitates seamless network-based communication between various independent applications (nodes) using a publisher and subscriber paradigm. Nodes can be written in either the C++ or Python programming languages.

Since it is open source and growing in popularity, drivers for many common sensors, such as the Xtion RGB-D camera, MicroStrain IMU, and Ublox GPS module have already been developed, facilitating rapid hardware integration. In addition, many state-of-the-art algorithms are available for straightforward integration.

On top of being an easy to use framework to facilitate robotics research, ROS provides a useful suite of tools for data collection and analysis. We cover a few of these highly useful tools that were used extensively in this research.

### rosbag

`rosbags` are data files that contain any of numerous ROS or custom data types along with time stamps for each data packet (message). Any message that is published on the ROS network can be recorded in a `rosbag`. The bag can then either be used for analysis, e.g. plotting data, or can be used to replay the data either in pseudo-real-time or at modified rates. When the bag is being played, nodes can subscribe to the data as if it was live.

This tool alone has saved countless hours during development and debugging. As an example to illustrate the significance of this tool, we were able to record visual odometry, IMU, and altimeter data during a manual flight. We could then replay the data in a lab setting and pass the data through our state filter to evaluate the performance of the estimator. By running the estimator offline but using real data, we were able to tune gains, catch bugs, and fix issues quickly and efficiently. This is true for many of the modules described in Chapter 3.

### rqt_bag

`rqt_bag` is another tool closely related to bag files, it allows you to graphically open a bag to analyze the contents. It provides a timeline of when messages are broadcast. The data of each message can either be viewed as raw data or can be plotted within the tool. Additionally, topics can be selectively published and the bag can be played and navigated with ease.

### rqt_graph

`rqt_graph` displays all the nodes and topics, that are currently active on the ROS network, in a graph. This allows the user to visualize and verify the data connections between nodes.

### rqt_plot

Data can be subscribed to and plotted real time using `rqt_plot`. Data fields within a message can be specified and visualized with a single command. This is a very valuable tool for real time assessment of system performance.

### tf

`tf` is a tool for relating coordinate frames to each other. For a single robot, numerous coordinate frames may be important, such as the body frame or camera frame. `tf` allows coordinate frames to be defined relative to one another in a tree structure. Then, rather than having to manually calculate the transformation between two coordinate frames, which can often be tedious or confusing, `tf` simply requires identifying the two desired frames.

### rviz

`rviz` is a 3D visualization tool. Many types of data can visualized using `rviz` including coordinate frames, point clouds generated from depth sensors like an RGB-D camera, vehicle paths, and even 3D CAD models of hardware. Additionally, `rviz` can be used for human input. For example, we use `rviz` as a primitive ground station to supply position goals to our aircraft.

## 2.4 Multirotor Simulator

We developed a multirotor flight simulator in ROS to enable software-in-the-loop simulation. The simulator is a functional replacement of the physical aircraft and includes nodes to simulate sensor measurements such as IMU, visual odometry, and altimeter.

The vehicle dynamics are simulated using the following set of equations that assume two axes of symmetry. The nomenclature follows directly from [20] where $c_x \triangleq \cos(x)$ and $s_x \triangleq \sin(x)$, $F_i$ is the force produced by the $i$th propeller, $\tau_i$ is the moment induced by the $i$th propeller, and $L$ is the arm length from the center of the multirotor vehicle to each propeller.

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{pmatrix} = \begin{pmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

$$\begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} rv - qw - g\sin(\theta) \\ pw - ru + g\cos(\theta)\sin(\phi) \\ qu - pv + g\cos(\theta)\cos(\phi) - \frac{1}{m}(F_1 + F_2 + F_3 + F_4 + F_5 + F_6 + F_{ge}) \end{pmatrix}$$

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)\sec(\theta) & \cos(\phi)\sec(\theta) \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix}$$

$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \frac{J_y - J_z}{J_x} qr + \frac{1}{J_x}(F_5 + F_6 - F_2 - F_3)L\sin(\frac{\pi}{3}) \\ \frac{J_z - J_x}{J_y} pr + \frac{1}{J_y}\left[(F_1 - F_4)L + (F_2 + F_6 - F_3 - F_5)L\cos(\frac{\pi}{3})\right] \\ \frac{J_x - J_y}{J_z} pq + \frac{1}{J_z}(\tau_1 - \tau_2 + \tau_3 - \tau_4 + \tau_5 - \tau_6) \end{pmatrix}$$

Although the current implementation uses hexacopter dynamics, the simulator is modular allowing for other platform dynamics, such as quadrotors or fixed-wing aircraft, by implementing a virtual function.

We use measured moments of inertia and mass from our hexacopter platform to define the simulation vehicle. An empirical model of ground effect is used in the simulation as well as the

enhanced drag model in [21]. The fidelity of our simulation is high enough that control gains that are tuned using the simulator work well on hardware with minimal modification.

Vehicle states are output from the simulator in multiple data formats, allowing for plotting and direct interfacing with other nodes. Additionally, `rviz` is used for a 3D visualization of the hexacopter, as shown in Figure 2.2.

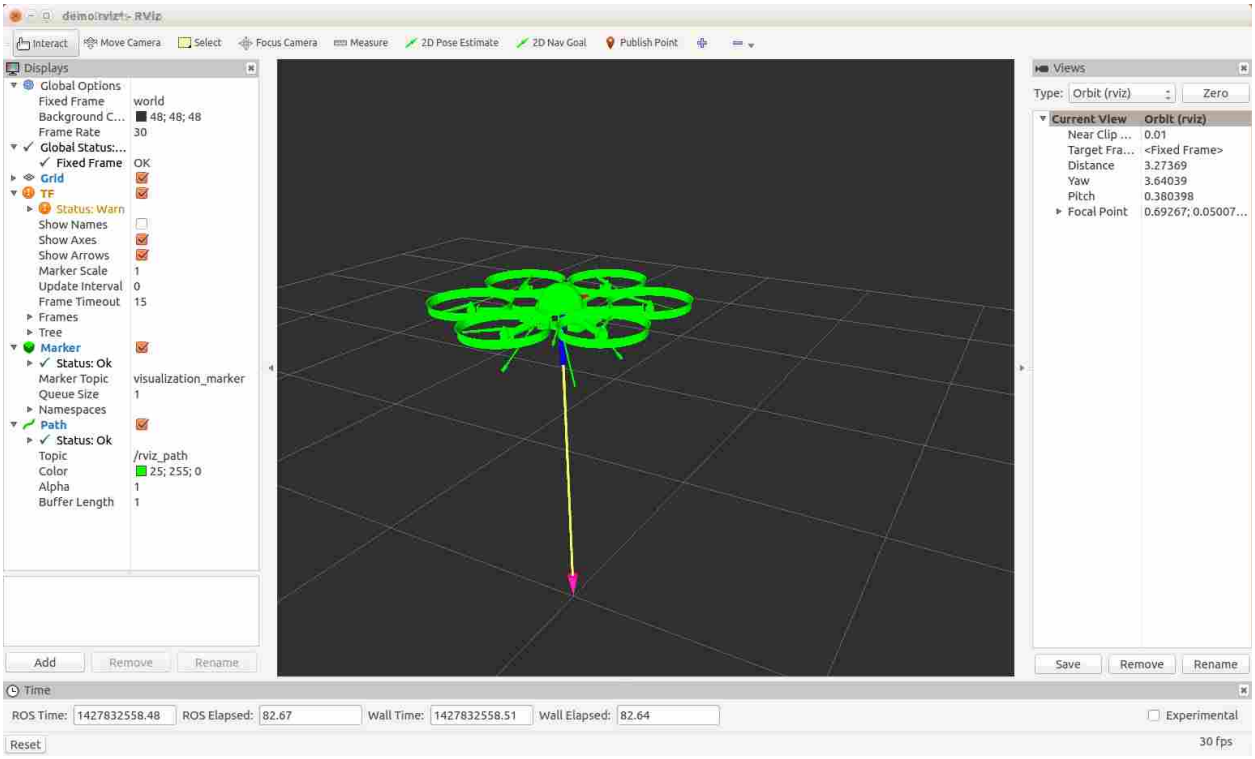



Figure 2.2: `rviz` visualization of the ROS multirotor simulator developed for software-in-the-loop simulation.

## 2.5 MATLAB Back-End Simulator

In addition to the flight simulator, a pose-graph back-end simulation was developed in MATLAB to verify incorporating GPS and robust least-squares in the back end. The back end is capable of handling real odometry measurements exported from ROS, or synthetic measurements. Odometry edges compounded to create a vehicle trajectory. Loop closures are added through proximity checks, and false-positive loop closure can additionally be added to random, or specific, pairs of nodes. Both valid and outlying GPS measurements can be added to the graph as well. We

have implementations of standard least-squares, switching constraints, and dynamic covariance scaling optimization, all of which feature Gauss-Newton iteration to converge on a solution.

The simulator has capabilities to generate graph files (.g2o) that can be parsed by the ROS implementation of g2o, as well as methods to load graph files from ROS into MATLAB. The implementation details of the simulation will be further discussed in Chapter 4.

# CHAPTER 3. RELATIVE NAVIGATION FRAMEWORK

The relative navigation framework [4], presented in Figure 3.1, is a novel system architecture developed to address the GPS-denied navigation problem by decoupling the flight critical processes from less urgent global updates. The framework is the confluence of a relative SLAM front end and a global back end.

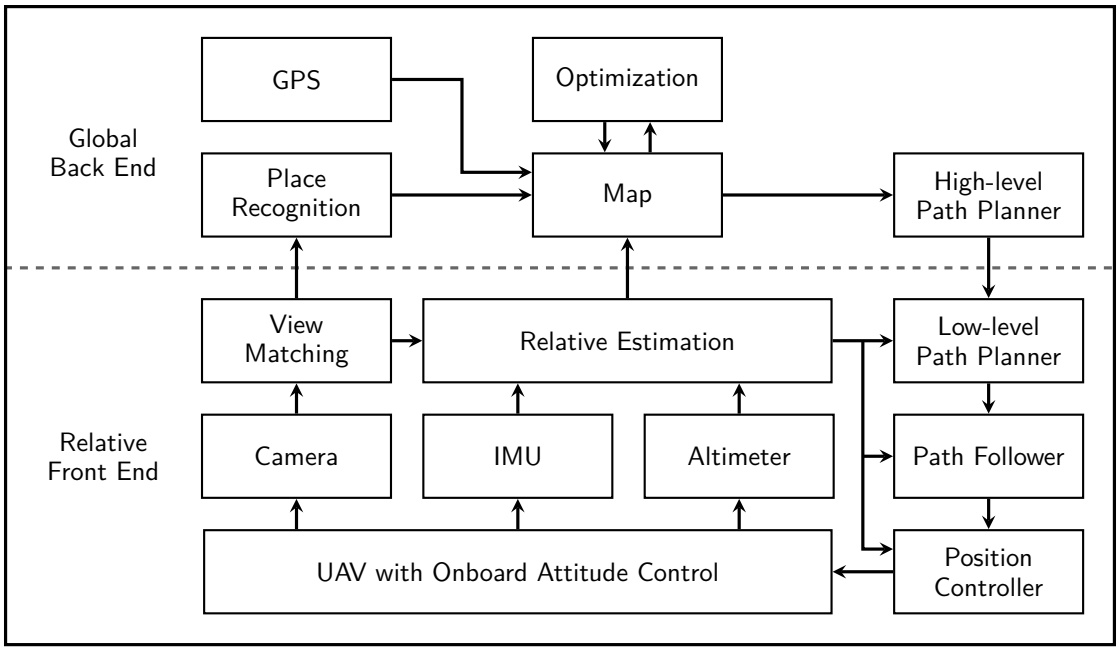



Figure 3.1: A graphical display of the relative navigation framework. It boasts a decoupled relative front end and global back end, allowing flight-critical processes to be executed without any ill-effects from global updates.

## 3.1 Framework Overview

The front end is responsible for fusing sensor measurements to generate odometry estimates suitable for pose-graph construction. Rather than estimating the global state of a vehicle, as

most navigation solutions do [13, 15, 17], this approach estimates vehicle states relative to locally established coordinate frames, entirely independent of global information. As the vehicles moves, new coordinate frames are defined and the estimate of the relative transformation between consecutive frames is output as an odometry edge for graph construction. The low-level path planner and follower compute paths to relative goals, provided from the global back end, and return desired relative states to a state controller. State estimation, path planning, and control all operate in the observable local coordinate frame.

Decoupled from the flight-critical estimation and control is the need for a global map. The back end constructs a map, represented as a pose graph, by compounding the odometry measurements provided by the front end. If additional loop closure edges are added to the graph, drift in front end estimates can be compensated in the global back end through graph optimization. Loop closures are added when the vehicle detects that it has returned to the same physical location that it has previously traversed. Loop closures over-constrain the graph and allow least-squares optimization to remove error induced by front-end drift. GPS measurements can also be directly applied to the graph to serve as constraints (Chapter 4). After the graph has been refined through optimization, yielding a globally consistent map, it can be used for various purposes, including high-level path planning. The only influence the global back end has on the front end is providing high-level goals. These goals may be adjusted through map optimization, but they do not directly affect front-end state estimates.

Motivating the development of the relative navigation paradigm are underlying issues with global state estimation and highly coupled frameworks, several of which are listed below.

- A UAV loses GPS signal and receives IMU measurements only for several minutes. Upon reacquiring GPS, the state estimate jumps drastically and the vehicle is unable to recover.
- A multirotor vehicle moves from indoors to outdoors. Upon acquiring GPS signal for the first time its global state estimates, with respect to an arbitrary origin inside the building, will jump drastically.
- After a loop closure, a UAV's current estimated global state may jump significantly resulting in sudden, unintentional, and unpredictable vehicle motion.

- After flying for some time, the size of the optimization problem delays any updates to the UAV's estimated global state. The vehicle's control suffers as a result.
- A vehicle receives an erroneous loop closure or GPS measurement. The estimated global state degrades without a method to later remove the effects of the outlying measurement.

The decoupled nature of our framework allows the global map to be optimized underneath the relative front end, allowing for continuous relative state estimates even when discontinuities in global estimates arise. While the global map may be contorted, the front-end relative state is unaltered.

An illustrating analogy of our framework is a person driving a car and following a map provided by a standard GPS navigation unit. Although the person is trying to follow a global path provided by the GPS unit, they do not blindly stare at the GPS while maneuvering through city streets. Rather, they look at their surroundings and navigate relative to their current view of the environment, focusing on safe driving; opportunistically looking at their map to localize themselves on the global path to inform which direction they need to go. By navigating relative to a known reference frame, even if there is uncertainty in the global position of that reference frame, global updates do not distract or derail the driver.

In the remainder of this chapter, we give an overview of the front-end modules used in our navigation scheme, following which we discuss the back-end modules. We then briefly cover some implementation details followed by results from indoor experimental flight tests.

## 3.2 Relative Front End

The components of the relative front end, introduced in Section 3.1 are explained in greater detail below.

### 3.2.1 Relative Multiplicative Extended Kalman Filter

The relative multiplicative extended Kalman filter (RMEKF) [22] is the crux of the relative navigation framework. It is a well known problem that without GPS updates, estimating a vehicle's global position and yaw states with relative measurements is analytically unobservable [23]. In

practice, however, these states appear to be somewhat observable, which over time leads to false confidence of the estimated global state. The RMEKF counters this problem by establishing and estimating its forward and right position $(p_x, p_y)$, and yaw $(\psi)$ states relative to local, gravity-aligned coordinate frames (nodeframes). Altitude above ground is observable when using a sonar or laser altimeter and is therefore estimated relative to the initial nodeframe. We assume constant height above ground which results in a 2D series of nodeframes.

After moving a short distance, or rotating a small amount, a new nodeframe is established by removing the roll and pitch angles from the current estimate of the vehicles state, aligning it with the gravity vector. Together, the resulting transform and its associated uncertainty form a 2D odometry measurement that is output to the back end. The filter then resets $p_x, p_y, \psi$, and respective covariances to 0. This process effectively transfers any state uncertainty to the non-critical back end, allowing the front end to maintain state certainty for its critical flight processes. The process is visualized in Figure 3.2.

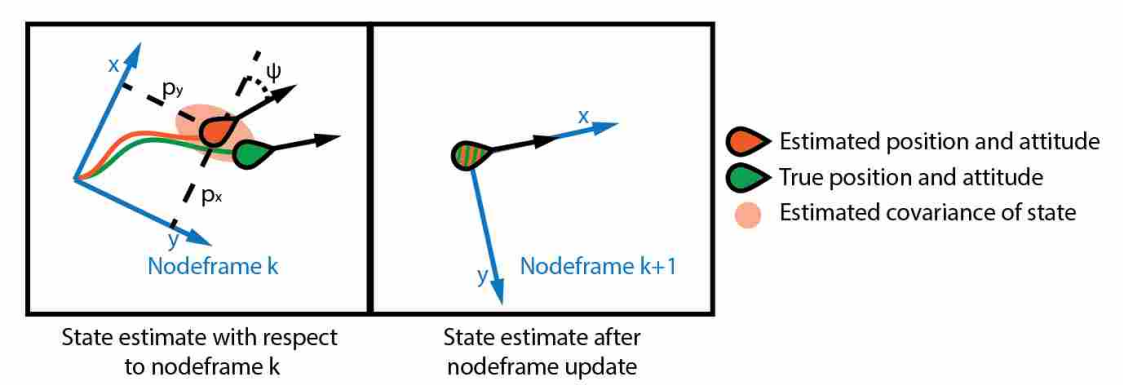



Figure 3.2: 2D illustration of estimated and true states relative to the current nodeframe. Position $(p_x, p_y)$ and yaw $(\psi)$ are defined relative to the current nodeframe, $k$. When a new nodeframe is declared, the estimated position and yaw are saved as an edge in the back-end pose graph. Nodeframe $k+1$ is defined as the current true state, which zeros the filter's estimate of $p_x, p_y$ and $\psi$ as well as the associated covariances. The state error and uncertainty are passed from the front end to the back end.

Quaternions are used to represent the vehicle's attitude for their computational advantage and to avoid the singularity known as gimbal lock that can arise when representing attitude with Euler angles. When using quaternions, the attitude error cannot be directly added to the current

state in the update step of the EKF. Instead, the attitude quaternion is multiplied by the error in attitude, earning the name multiplicative EKF.

In our implementation, the filter's underlying dynamics make use of both the typical kinematic relationships [20] as well as an enhanced drag model [21] where velocity estimates are constrained using both accelerometer measurements and drag terms that are proportional to the body-fixed forward and right directions. Mechanization is employed to use gyroscope measurements as inputs to the estimator, while accelerometers are used as measurement updates. Additionally, a modified version of the Depth Enhanced Monocular Odometry (DEMO) algorithm [24] is used to provide egomotion measurements.

Figure 3.3 compares the forward and yaw states from the RMEKF to truth as provided from a motion capture system. The discontinuities arise when new nodeframes are established.

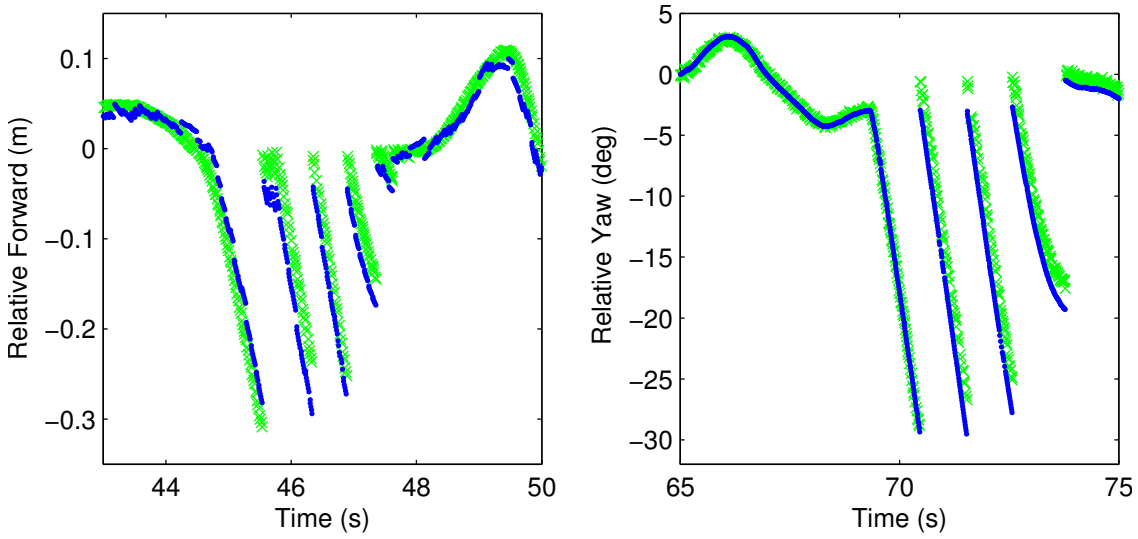



Figure 3.3: RMEKF performance plots of the forward direction position and yaw angle estimates (blue) compared to truth (green). The discontinuities indicate a change in nodeframe during continuous backward and counterclockwise motion.

### 3.2.2 Visual Odometry

Visual odometry (VO) is the process of computing a camera's egomotion by comparing two images. There are two general categories of VO algorithms, appearance-based and feature-based. Appearance-based VO uses pixels from the entire image to compute the transformation between

two frames. Feature-based approaches, on the other hand, down sample the image by extracting visually distinct features, such as corners, that are tracked between images.

VO has been performed using both stereo and monocular cameras. More recently, RGB-D sensors have become increasingly popular sensors for VO. In addition to an RGB image, RGB-D sensors capture a depth image that encodes a distance measurement for each pixel of the RGB image.

Using a monocular camera, rotations can be recovered by VO, but translations can only be computed accurately up to an unknown scale factor. This problem is referred to as scale ambiguity and arises when there is no measure of depth, or distance, to the image. Stereo and RGB-D cameras provide the depth information necessary to resolve the scale factor and accurately compute translation.

Another distinction among VO algorithms is the difference between consecutive frame and keyframe approaches. Consecutive frame algorithms, as the name implies, use consecutive camera images to compute the motion of the camera. Keyframe approaches are able to use non-consecutive frames to compute transformations. They declare an image as a keyframe and compute the transform between it and all subsequent images until a new keyframe is declared. Various methods can be used to decide when a new keyframe should be declared including thresholds on distance or rotation from the keyframe. The advantage of keyframe-based algorithms is that they are free from temporal drift errors [23].

A useful compromise between both methods is known as bundle adjustment. In addition to the transformations between consecutive frames, bundle adjustment computes and uses the transformations between all, or a subset, of nonconsecutive frames. An optimization occurs and returns the most probable transformation from the first to last frames used.

Keyframe-based approaches are especially fitting for use with the RMEKF. New keyframes can be established at the same time as nodeframes. This makes VO updates directly relative to the last nodeframe. For many of our preliminary flight experiments, we used an implementation of the feature-based keyframe VO algorithm outlined in [4].

Although keyframe-based algorithms are particularly fitting to the relative navigation framework, they are not requisite. For the flight experiments presented later in this chapter and in Chapter 4, DEMO, a consecutive frame algorithm, is adapted to work in the relative navigation framework.

Perhaps the most noteworthy advantage of DEMO over other state-of-the-art algorithms is its ability to estimate image feature depth when a depth measurement is unavailable. This allows RGB-D based platforms to perform VO outdoors, where depth information is saturated by infrared scatter from the sun or when features are outside the depth sensor's range.

DEMO's natural output is a global estimate of the camera's pose in a left-up-forward coordinate frame. It outputs a low-rate (1 Hz) bundle-adjusted transform using a subset of the previous several frames. High-rate transforms are provided at approximately 16 Hz. We modify DEMO to establish a pseudo-keyframe after each bundle adjustment. The relative transformation between the keyframe and the following camera poses is computed and converted into the more typical right-down-forward camera frame that the RMEKF expects. Using this approach, we are able to seamlessly use a global, consecutive-frame VO algorithm, highlighting the easily adaptable and modular framework.

We refer the interested reader to good tutorials on visual odometry found in [25, 26] and to [27] for a comparison between various visual odometry algorithms.

### 3.2.3 Control

Another flight-critical process, related to state estimation, is state control. The area of multirotor state control has been widely explored in the literature. Conventional approaches to rotorcraft control rely on global state estimates. In a relative navigation system, since state estimation and control occur in the local nodeframe, it is essential that these modules remain synchronized. If somehow, either due to latency or dropped communication packets, the commands and state estimate coordinate frames become out of sync, we have a failsafe to hover at whatever state the vehicle believes it is in; the origin of the current nodeframe becomes the commanded state, until the mismatch is resolved. In practice, unsynchronization rarely occurs, and lasts less than several hundredths of a second when it does, not impacting performance.

Figure 3.4 is a block diagram of our general control structure. The architecture is quite flexible, allowing one to easily implement new state controllers by redefining a few virtual functions.

Although several controllers have been designed and implemeneted throughout the course of this research, including PID position, sliding mode, and adaptive altitude, we primarily use two

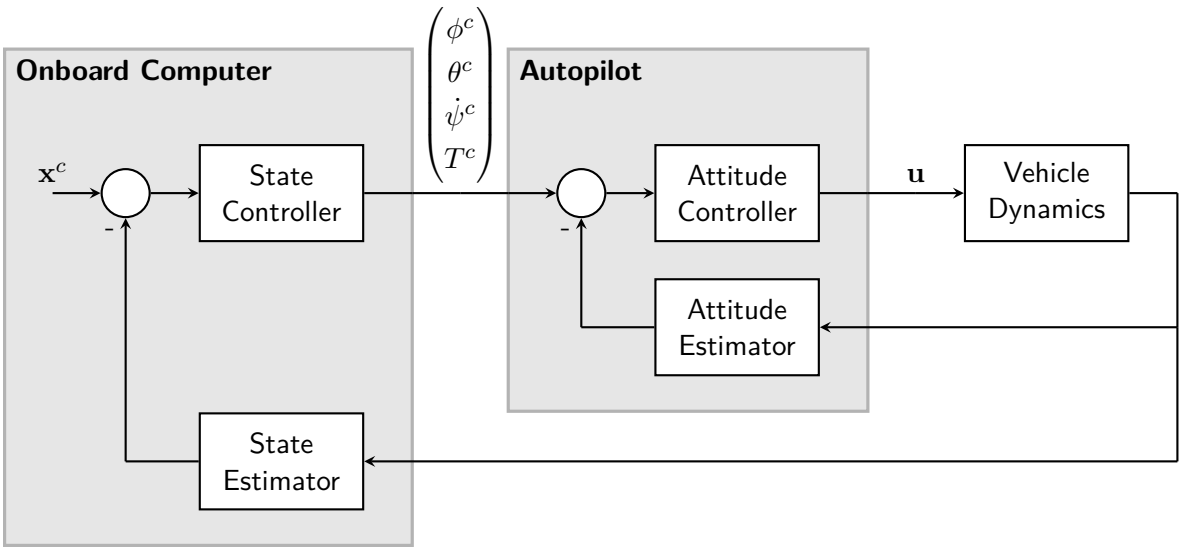



Figure 3.4: A diagram describing the control architecture used in our approach. An onboard i7 computer hosts a state estimator (RMEKF) and state controller (currently a body-fixed velocity controller). Outputs from the state controller are commanded roll and pitch angles, thrust, and yaw rate ($\phi^c, \theta^c, \dot{\psi}^c, T^c$). These commands are used by the autopilot for attitude estimation.

controllers for this research, a differential flatness position controller and a body-fixed velocity controller.

The differential flatness controller is an implementation of the controller detailed in [28]. This controller allows trajectory-based aggressive maneuvers. However, for our experiments, we have no desire to execute aggressive maneuvers, making differential flatness unnecessary for our problem.

Instead, a body-fixed velocity controller is used to focus on smooth, non-aggressive control. The velocity controller was designed and used for the results in Section 3.5 and Chapter 4. The control inputs are desired body-fixed velocities in the forward and right directions, altitude, and yaw-rate. We use PID loops on the desired velocities and altitude to compute acceleration commands in the respective directions. Model inversion is subsequently used to calculate the desired roll and pitch angles and thrust. Thrust is mapped to a desired throttle command using an empirically derived linear mapping. The roll, pitch, and throttle commands are then passed to the onboard autopilot to perform low-level attitude control. Desired yaw rates are passed directly from the velocity controller to the autopilot attitude controller.

The practical advantages of using the body-fixed velocity controller opposed to a position controller, such as the differential flatness or PID controllers, is highlighted in the following section.

### 3.2.4 Path Planning and Following

Low-level path planning and following are critical modules for autonomous navigation. In the relative navigation framework, the high-level path planner is responsible for transforming a global goal into the coordinate frame of the most recently established nodeframe while the low-level path planner generates a path to the transformed goal. Mature path planners should incorporate obstacle avoidance methods to identify and plan paths around potential obstructions.

Throughout this research, we have implemented and tested several low-level path planners and followers, including a straight line planner featuring 2D voxel-grid costmap obstacle avoidance, and the waypoint follower outlined in [20]. However, to simplify our platform, for the results in this thesis, we bypass the path planner and follower by providing body-fixed velocity commands to our velocity controller from a wireless Xbox controller. An operator acts as a functional replacement of an obstacle avoidance capable path planner and follower. Although this simplification does remove some level of autonomy, it allows us to move focus from front-end flight capabilities to back-end map construction and maintenance.

## 3.3 Global Back End

Decoupled from the flight-critical front end is the need for a global back end for global state reconstruction. The following details each element of the global back end.

### 3.3.1 Pose Graph

To maintain a spatially, or globally consistent map, the back end stores information about the vehicle's trajectory as a pose graph. A pose graph is a graphical representation of a vehicle's path that encodes global vehicle pose estimates as graph vertices and the relative transformation between two poses as graph edges. A pose graph representation is effective because efficient graph optimization algorithms have been developed that can be applied to refine pose estimates.

Additionally, the graph conveniently serves as an abstract map that can be used for 3D human-readable map construction as well as high-level path planning.

Figure 3.5 depicts how a pose graph is constructed from vehicle odometry. Graph vertices, $x_i$, represent the global estimate of a nodeframe's translation and orientation. The graph edges, $z_i$, connecting two consecutive poses encodes the relative transformation between two nodeframes and its respective covariance. Where typical SLAM back ends are provided with global nodes and compute the edges between consecutive nodes, the relative navigation framework provides odometry edges directly. A new nodeframe is defined by the odometry measurement from the previous nodeframe. In this framework, it is natural to use the uncertainty associated with the odometry measurements as the edge covariance. The transformations encoded by each edge are compounded with all previous transformations to reconstruct the global position of the new nodeframe and a corresponding vertex is added to the graph. Despite the subtle difference in graph construction, both graphs can be treated in a unified way after construction.

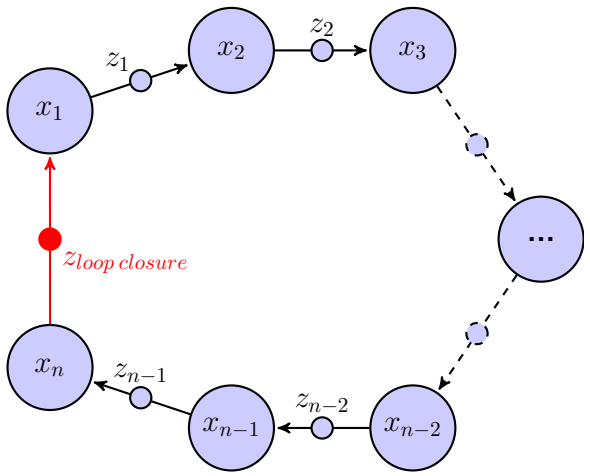



Figure 3.5: Example graph constructed from odometry and loop closure edges.

A pose graph is also capable of encoding measurements other than odometry. Loop closure measurements introduce edges between two existing, non-consecutive nodes (Section 3.3.2). Similarly GPS measurements can be added as edges between the vertex where the measurement was received and the origin of the global coordinate system (Chapter 4). Since Lu and Milios's seminal

work in [8], pose-graph SLAM has become the predominate method of SLAM in the literature. An excellent tutorial on pose-graph SLAM is found in [9].

In our implementation, we construct a 2D pose graph using the popular g2o library [18]. g2o has been developed as a general framework for graph optimization. It leverages efficient methods and algorithms for graph construction and optimization. We use a 2D graph rather than 3D because the nodeframe-to-nodeframe odometry measurements provided by the front end are 2D $(p_x, p_y, \psi)$. This is because roll and pitch angles are removed from nodeframes by definition and we assume flight at a constant altitude above ground.

### 3.3.2 Place Recognition

A vehicle's global position estimate will drift significantly over time in the absence of global updates. A common solution in the SLAM literature makes use of place recognition. In place recognition, the current image is compared with previous images to determine if the vehicle has returned to a previously visited physical location. While a naive solution would require significant computation and memory, not generally available with the SWAP constraints of a small UAV, the computer vision community has developed efficient vocabulary-based place recognition algorithms. As with VO algorithms, distinct image features are represented by mathematical descriptors. Using a large, representative training dataset of images, the most prominent, distinct feature descriptors are saved offline and referred to as words in a vocabulary. During flight, any image can be succinctly represented by the set of nearest vocabulary words found in the image. Images can be quickly compared using word occurrences, similar to many search engine algorithms [29]. The approach of using words and a vocubulary to describe images is referred to as a bag-of-words approach. Further work has improved place recognition performance in the presence of aliasing, where high correlation is found on non-correlated images, like pictures of brick walls [30].

We use a ROS implementation of the FABMAP algorithm called OpenFABMAP [10]. The structure of our place recognition implementation follows the framework presented by CyPhy Lab [31]. Each keyframe image is passed through the place recognition software and archived with a description of the nodeframe that it corresponds to and the node to body coordinate frame transformation, $T_{\mathrm{N}}^{\mathrm{B}}$, when the keyframe was taken. The translation component of $T_{\mathrm{N}}^{\mathrm{B}}$ is zero be-

cause the node and body frames are collocated. After a match is found, the images are recalled and visual odometry methods provide the estimated transformation from one nodeframe to another. An example image pair and corresponding transformation is shown in Figure 3.6. The transform is communicated to the pose graph and added as an edge, $z_{loop\,closure}$ (Figure 3.5), between the non-consecutive nodes, allowing for optimization and reduction of accumulated drift.


Figure 3.6: A superimposed image pair that was returned as a match by OpenFABMAP. A visual odometry algorithm computes the transform between the two images and returns it to be added in the pose graph. The transformation is visualized as the green lines overlaying the image.

We include Figures 3.7 and 3.8 as an implementation note. Figure 3.7 shows the four ROS nodes that are primarily responsible for maintaining the pose graph. The pose graph is constructed in hex_map. As keyframes are declared, the images are stored in the rgbd_cache node with their associated node info consisting of the node-to-body transformation and nodeframe identification number corresponding the keyframe. openfabmap also receives keyframes as they are declared, extracting and storing only a description of the image using a bag-of-words approach. The description of the current image is compared to all other previously acquired images. When it finds a strong correlation between two images, it sends the two associated keyframe identifiers to geometry_check. This node requests the two corresponding images and node info from rgbd_cache. geometry_check estimates the relative transformation between the two images, $T_{\mathrm{CA}}^{\mathrm{CB}}$, with Leishman's

feature-based visual odometry algorithm described in [4]. The transformation $T_{\mathrm{NA}}^{\mathrm{NB}}$ is computed using the node-to-body transforms acquired from the node info database. Figure 3.8 illustrates the series of transforms used to estimate node-to-node transformations and is summarized by Equation 3.1, where $(\cdot)^{-1}$ is the inverse transform operator.

$$T_{\mathrm{NA}}^{\mathrm{NB}} = (T_{\mathrm{NA}}^{\mathrm{BA}})(T_{\mathrm{C}}^{\mathrm{B}})^{-1}(T_{\mathrm{CA}}^{\mathrm{CB}})(T_{\mathrm{C}}^{\mathrm{B}})(T_{\mathrm{NB}}^{\mathrm{BB}})^{-1} \tag{3.1}$$

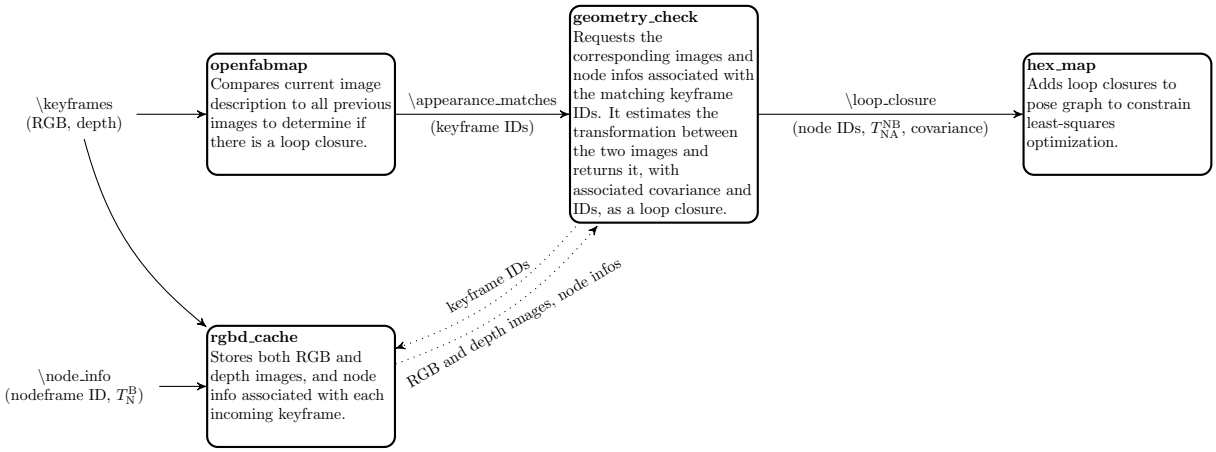



Figure 3.7: Graph showing the interactions and responsibilities of the four primary back-end nodes. rgbd_cache stores required information, including RGB and depth images, and node-to-body transformations, for each declared keyframe. openfabmap compares keyframe images to detect loop closures. When a loop closure is detected, geometery_check retrieves the associated images from rgbd_cache and computes the relative transformation between the two associated nodeframes. The transformation is added as a loop closure between corresponding nodes in hex_map.

### 3.3.3 Map Optimization

Because of erroneous odometry measurements, graph optimization is critical to maintaining a globally consistent map. A key advantage of using a pose graph to represent a map is that it can easily be formulated as a least-squares optimization problem where poses and edges become free variables and constraints respectively.

Least-squares optimization attempts to find the most likely arrangement of poses given a set of odometry, loop closures, GPS, or other measurements. This is accomplished by minimizing a cost function, where each edge contributes a quadratic cost. Many efficient graph optimization

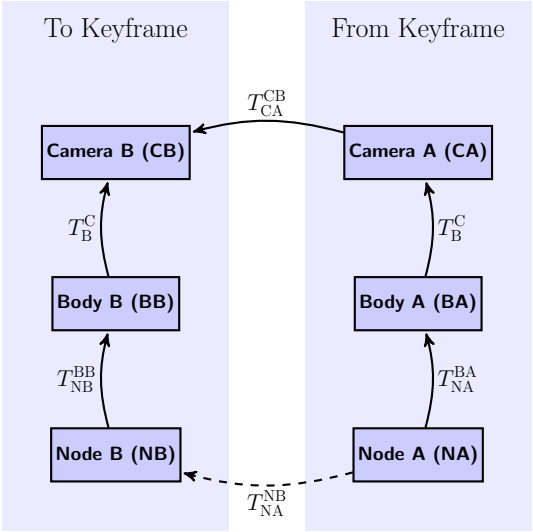



Figure 3.8: A visual description of the relationship between various coordinate frames. The goal of place recognition is to compute the transformation from Node A to Node B, $T_{NA}^{NB}$.

algorithms exist, including iSAM [16], $\sqrt{\text{SAM}}$ [32], TORO [14], and g2o [18], some of which are popular open-source projects [12].

Intuitively, pose-graph optimization can be thought of as a mass spring system settling to a state of least energy. In this analogy, graph vertices are masses and edges are springs. A spring's stiffness corresponds to the edge's covariance, becoming increasingly stiff as the covariance approaches zero. In a thought experiment, one can visualize that when a graph constructed of only odometry edges is optimized, the output is simply the original graph because there are no additional loop closure springs to deform the system. When loop closures are added to the graph, additional springs are added between non-consecutive masses, resulting in a graph that more closely represents the true vehicle trajectory. During optimization, the springs with the lowest stiffness (high uncertainty) will be distorted the most.

For the graph optimization results presented in this chapter, we use a ROS implementation of the g2o libraries. We use ten iterations of the Gauss-Newton algorithm unless otherwise stated. We refer the interested reader to Appendix A for an outline of how the least-squares optimization problem can be formulated from a pose graph.

### 3.3.4 High-Level Path Planning

The purpose of the high-level path planner is to provide global goals to the relative front end. Goals can be provided by any number of methods, such as human input or exploration algorithms. In the relative navigation framework, the high-level path planner uses the current estimate of the vehicle's global state, provided by the pose graph, to compute a path to the goal in the inertial frame. In the complete absence of GPS, this inertial frame is the initial vehicle pose, $x_0$. When GPS is available, even sparsely, the inertial frame is the origin of the local UTM zone, or other global coordinate system (see Chapter 4). High-level path planning has been widely explored in the literature resulting in numerous methods of generating paths to achieve desired characteristics, such as shortest distance or minimized threat exposure. Our framework allows for any of these path planners to be used in conjunction with a low-level path planner to transform the global path into a local path relative to the most recent nodeframe.

For the following experiments we do not incorporate a high-level path planner. To limit the scope of our experiments, emphasizing estimation, control, and back-end optimization, we circumvent the need for a path planner by supplying body-frame velocity commands directly to the controller.

## 3.4 Implementation

Each of the modules presented above is implemented as an independent ROS package containing one or more nodes. By using ROS, classes, and a common design, we have made the implementation of this framework highly modular. We are able to swap virtually any module in the system for an alternate, provided the data interfaces are common. For example, we are able to incorporate DEMO as a plug and play replacement for Leishman's VO algorithm [4] with a wrapper function that organizes the data into compatible message types.

## 3.5 Results

Experimental flight tests were conducted to validate the framework and implementation described above. The following results are from a flight test conducted on the third floor of BYU's Wilkinson Student Center (WSC). The objective of the flight test was to complete loop closures

by traversing the hallways in Figure 3.9 and optimizing the resulting graph to reconstruct a globally consistent map. All estimation, control, place recognition, and optimization were completed onboard the aircraft in near real time.

Traversing the WSC hallways is quite challenging for the following reasons.

- Several stretches of the hallways are plain walls and relatively featureless, making visual odometry and loop closure estimation challenging.
- The south wall of the most southern hallway has many large, glass, reflective windows. Visual odometry methods are not robust to reflective features and this creates a challenging environment to estimate egomotion.
- The width of the outer and inner hallways are 2.5 meters and less than 2 meters wide respectively. The diameter of the hexacopter is approximately 1 meter. This leaves very little room for error, particularly in the inner hallway.
- The limited hallway width constrains prop wash airflow inducing significant ground and wall effects. Recessed doorways make these effects unpredictable and difficult to model.

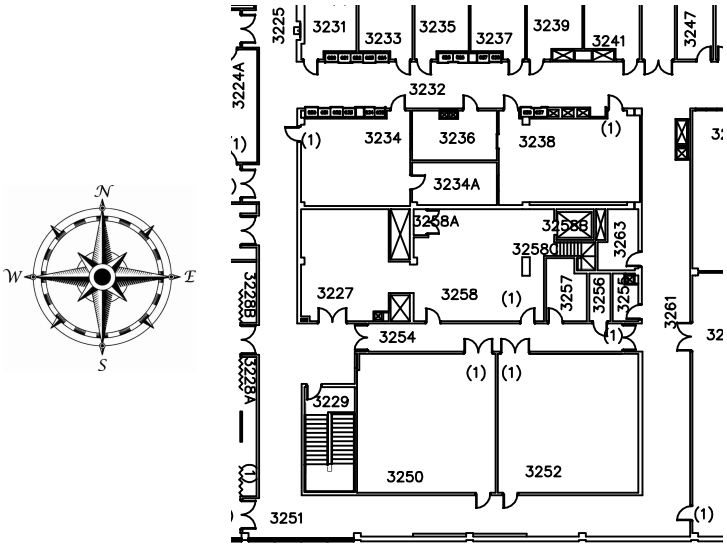



Figure 3.9: The third floor Wilkinson Student Center floor plan where indoor flight tests were performed. Loops are flown around the outside loop as well as the upper and lower loops.

As a documentation note, preceding the experiment we submitted our flight plan to BYU Risk Management and campus scheduling for approval. Additionally, we alerted the BYU police both prior to the experiment and after completion. Example documents that were submitted to risk management and campus scheduling can be found at http://magiccvs.byu.edu/wiki. We only operated after WSC closing hours, provided safety personnel to direct traffic around the experiment, had a faculty member present during the test, and had an experienced safety pilot (Figure 3.10) to override control in the event of unexpected vehicle behaviour.

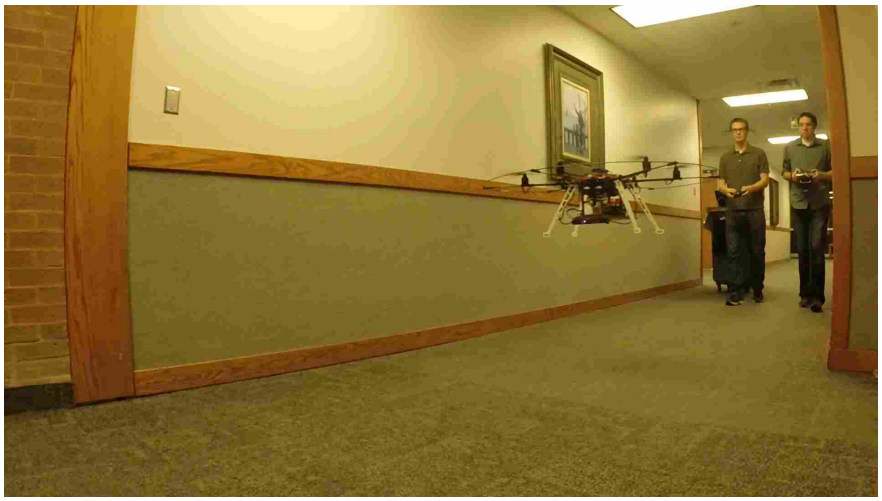


Figure 3.10: Our multirotor experimental platform being flown down the hallways of the WSC. The operator on the left gives body-fixed velocity commands to the hexacopter via a wireless Xbox controller. The operator on the right is the safety pilot who can override the state controller by providing roll, pitch, thrust, and yaw rate commands via a RC transmitter.

Using the relative navigation framework implementation outlined above, we were able to complete a 9-minute flight, controlling hexacopter states based on estimates provided by the RMEKF, forming five distinct loops while both constructing and optimizing a global map of the traverse.

Figure 3.11 highlights the efficacy of the relative front end. The visual odometry measurements provided by the DEMO algorithm form a good estimate of vehicle motion (left), however we do see a significant amount of drift, particularly when turning corners. This motivates the need for a sensor fusion filter to leverage both high-rate information from the onboard IMU, and

the relative low-rate visual odometry. The graph formed using the RMEKF is the fusion of these measurements and much more closely resembles the actual path traversed by the aircraft (right).

To be clear, the VO and RMEKF odometry measurements are relative to the previous nodeframe (indicated by blue dots). The measurements are compounded here to show unoptimized global state reconstruction. Neither VO nor the RMEKF ever believed the vehicle was flying through a wall. The system allows the current state to reside ideally with respect to its local environment at the expense of shifting the global map around the local frame. This can be thought of as the map traversing through the vehicle's relative coordinate frame rather than the vehicle passing through a stationary map.

Throughout the flight the safety pilot never provided input to the system, showing that the RMEKF provided smooth and accurate relative state estimates suitable for real time control of vehicle states. Although the front end performed admirably throughout the experiment, from the figure it is evident that there is substantial drift in the global state estimates; however, we were able to successfully constrain error and maintain a globally consistent map by closing loops and optimizing the back end map.

In Figure 3.12, we see the final pose-graph constructed from front-end odometry (blue) as well as loop closures (red) detected by place recognition software before optimization (left). Although our implementation of place recognition only compares the images captured at each nodeframe declaration, we were still able to detect more than 150 loop closures without a single misdetection (a topic of discussion in Chapter 4). After batch optimization (right), the map is consistent with the experimental trajectory.

We note that during every pass of the south east corner of the WSC, visual odometry would provide less accurate estimates due to sparse, coplanar image features, while providing the same confidence on the estimates. This results in a slightly skewed optimized map, as seen on the bottom leg of the graph.

To illustrate that large jumps in global state estimates, for example after optimization, do not affect the front end in any way, we ran the back end real time, onboard the aircraft. After each loop closure, ten iterations of Gauss-Newton optimization were run resulting with Figure 3.13. Despite repeatedly running optimization on the aircraft in real time, front-end performance was not perceivably degraded. The unoptimized bottom leg of the graph is a consequence of turning off

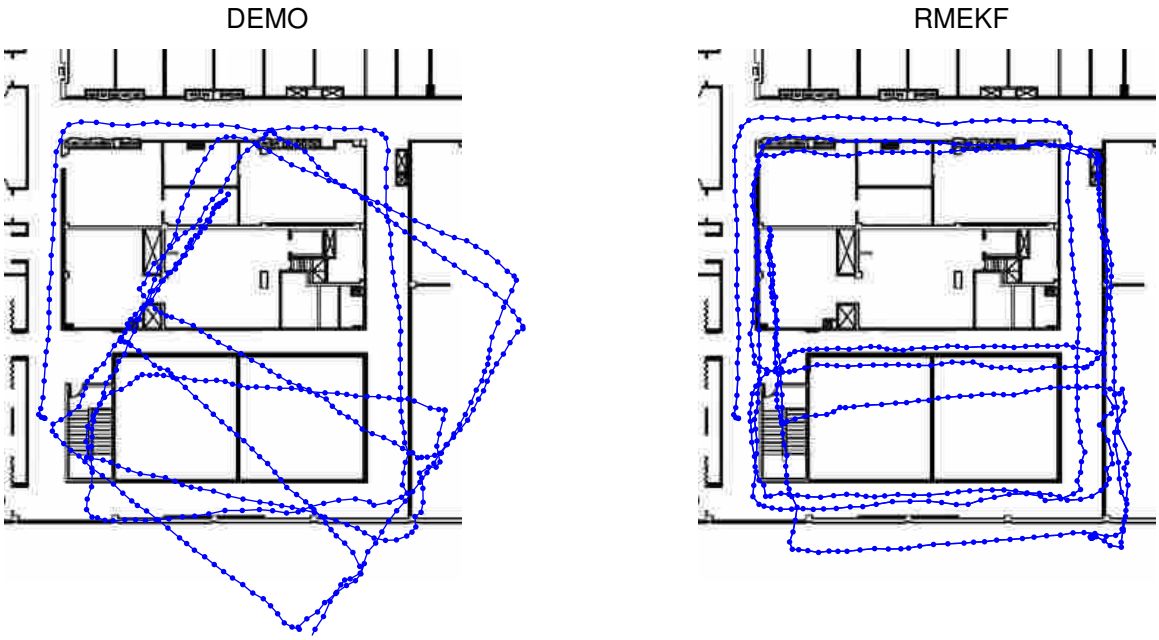



Figure 3.11: Comparison of the graphs constructed strictly from visual odometry mesaurements (left) and from RMEKF odometry measurements (right) where the blue dots and lines represent graph nodes and edges respectively. Although DEMO performs quite well, it is clear that the RMEKF significantly enhances state estimation.

the back end before it could complete optimization for all received edges. This could be alleviated by doing batch optimization at opportunistic intervals.

Post-process batch and incremental optimization are compared in Figure 3.14. Although the same graph is being optimized, in other words the same constraints between corresponding nodes, the results between batch (left) and incremental (right) are slightly different. The differing solutions result from slightly different initial node conditions. These slight changes in initial conditions during incremental optimization lead to different local minima than when batch optimization is run. Although the effects of local minima are barely observable in this instance, we show a more obvious example in Chapter 4.

## 3.6 Conclusion

In concluding this chapter, we impress that the relative navigation SLAM framework is a viable solution to the GPS-denied problem. The front end is able to operate completely independent of the back end. Discontinuities in global state estimates do not have any ill effect on the front

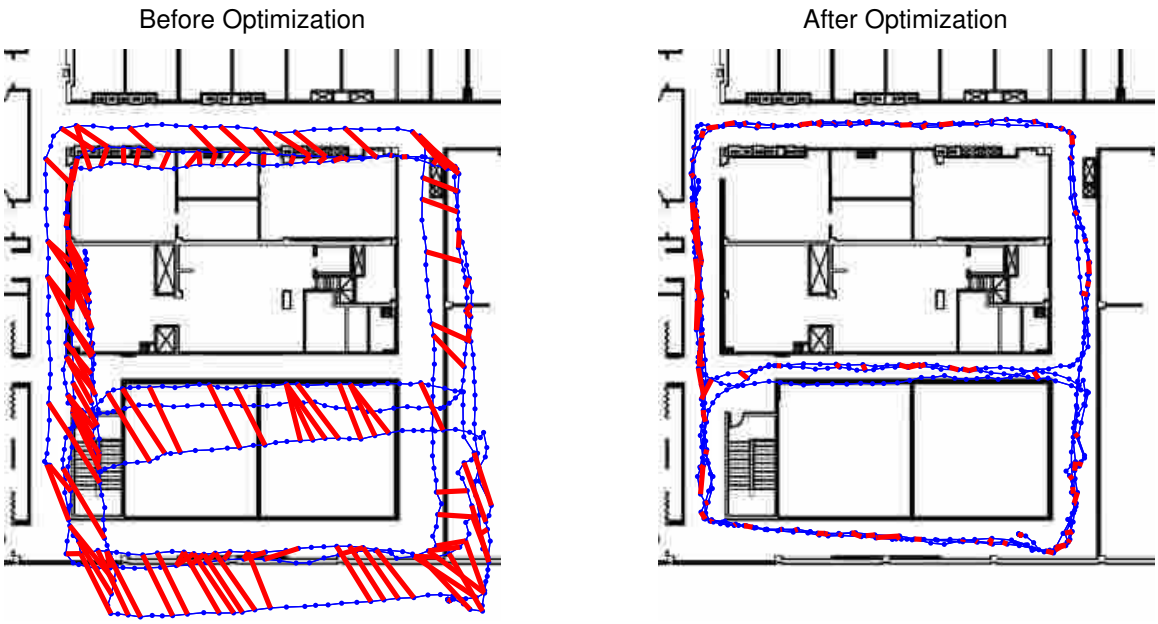



Figure 3.12: The unoptimized graph (left) with detected loop closures (red) compared to the optimized graph (right). The optimized result is clearly a better representation of the flight path.

end's ability to estimate and control states. Further, we have shown that the front end's relative states estimates can be leveraged to construct and optimize a global map, either in real time or post processed, onboard or offboard.

In these experiments, we circumvent the need for a path planner by providing body-fixed velocity commands by an operator. In the future, we plan on reimplementing position control and removing the need for an operator. Closing the loop from the back end to the front end via a high-level path planner is another anticipated area of immediate future work.

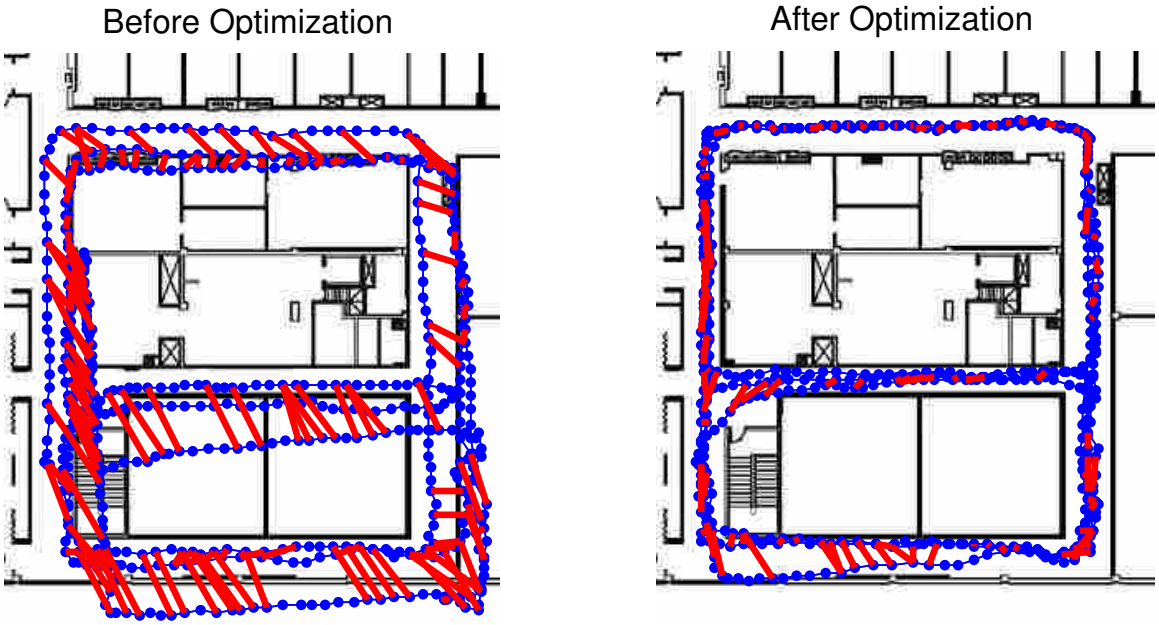



Figure 3.13: Comparison of the unoptimized graph (left) and the incrementally optimized graph (right) that was generated in real time, on board the aircraft.

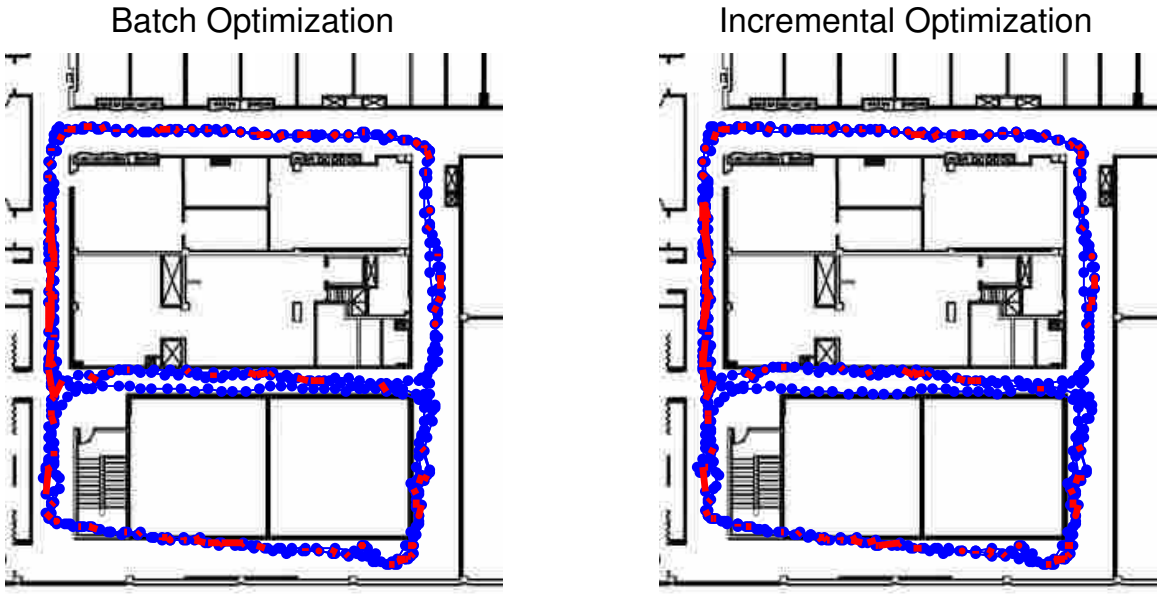



Figure 3.14: Slight discrepancies between batch optimization and incremental optimization arise from varying the initial node conditions. Despite having the exact same constraints, the variation in initial conditions result in different local minimum solutions.

# CHAPTER 4. INTERMITTENT GPS IN GLOBAL BACK END

In Chapter 3 we described the relative navigation framework as a solution to GPS-denied navigation and demonstrated it indoors on a multirotor aircraft. The results show that we are able to construct a graph from front-end odometry measurements as well as optimize it to create and maintain a globally consistent map. Although it is impressive to navigate in the absence of GPS, the described approach does not take advantage of any potentially available GPS measurements. Additionally, it provides no robustness to outlying measurements.

In this chapter, we develop an approach to leverage intermittent GPS as direct measurement updates in the global back end. Methods of making least-squares map optimization robust to outliers are discussed. We address two issues that arise when incorporating GPS measurements, first concerning the scale of GPS errors, and second regarding the initial guess of graph orientation. Finally, we give results for simulated, hand-carried, and flight test datasets, demonstrating the ability to maintain globally consistent and aligned graphs in the presence of false-positive loop closures and intermittent and degraded GPS.

### 4.0.1 GPS in Front End

In Section 1.2.1 we discussed graph-based SLAM, concluding with several impressive examples of autonomous UAS solutions that address GPS-denied navigation [13, 15, 17], and in Chapter 3 our own approach is detailed. However, because these systems assume a GPS-denied environment, they do not utilize GPS measurements even when they are available.

A common approach to incorporating GPS in a SLAM framework is to use GPS measurements in front-end estimation. Shen et al. [23] use a UKF filter to fuse GPS with inertial and exteroceptive measurements, handling both delayed and out of order measurements. They demonstrate their front end transitioning between indoors and outdoors to show that they are able to maintain smooth estimates even during periods of GPS drop outs.

Similarly, Chambers et al. [33] use a UKF to fuse sensor measurements emphasizing smoothness. They use a two stage gating process on sensor measurements, including GPS. First, they use the sensor's reported error estimate and reject the measurement if the error exceeds a defined threshold. If the measurement passes the first gate, a $\chi^2$ test is performed. The $\chi^2$ test evaluates how closely the predicted measurement and actual measurement are related. If the $\chi^2$ test exceeds a designated threshold, the measurement is rejected, but is stored for later use in the event that it becomes more supported after receiving more measurements. Their method exhibits smooth estimates that are suitable for control.

Both Shen and Chambers are able to handle intermittent GPS, Chambers even rejects outlying GPS measurements, but both methods rely exclusively on the front end to detect and eliminate such errors. Additionally, these methods are not suitable for our relative front end because our states are defined relative to local coordinate frames. With highly sparse GPS measurements, it would impossible to accurately represent absolute GPS measurements as relative measurements in our framework. This problem will be further addressed in Section 4.1.

### 4.0.2 GPS in Back End

In the ground robotics community, Rogers et al. [34] use the back end to evaluate incoming GPS measurements. Their method evaluates the norm of the distance between the last two GPS measurements and the distance between the corresponding two nodes where the GPS measurements were received. If the norm is within 10% of the greater distance, the new GPS measurement is accepted and used in the front end estimation. Their method explores using the back end to validate GPS measurements for use in the front end, but still relies on being able to apply sparse, absolute measurements in their front end.

Rather than using GPS in a front-end filter, Rehder et al. [35] apply GPS measurements directly in the back-end graph. They employ a virtual zero node that behaves as the origin of a absolute coordinate system (e.g., longitude and latitude or UTM). By adding the virtual zero, both relative measurements (odometry) and absolute measurements (GPS) can be treated identically in graph construction and optimization. In their experiments, they traverse 2 km on a pontoon boat. In the 12 minute traverse, they only receive six interspersed GPS measurements but are able to reconstruct a map within a mean of 5 m accuracy. They do not report using loop closures

to further constrain the graph. The advantage of the virtual zero is that the front end can be completely decoupled from GPS. However, without any front end filtering, blindly accepting GPS measurements in the back end is prone to outliers. A single outlier in a least-squares optimization problem can catastrophically derail the solution.

### 4.0.3 Robust Least-squares

In recent years there have been significant advances in making SLAM back-end optimization robust to outliers. Morton et al. [36] characterizes GPS sensors with richer noise models. They argue that outlying measurements arise from poor error models due to the assumption that edge error is characterized by a zero mean Gaussian. Rather than using the common approach of using a unimodal Gaussian to represent an edge's uncertainty, they propose using a max-mixture Gaussian model. Using max-mixture models, outlying measurements induce minimal cost in optimization, resulting in globally consistent maps.

Another approach to robust least-squares optimization is the Realizing, Reversing, Recovering (RRR) algorithm, proposed by Latif et al. [37]. It rejects outlying loop closures by performing consistency checks throughout the graph. They group topologically related loop closures into clusters. Once a cluster is formed, they perform an intra-cluster consistency check. This is an optimization involving only the loop closures within the cluster. If the $\chi^2$ error of the whole cluster exceeds a threshold, then the entire cluster is rejected from future optimization. If the cluster is within the threshold, each of the loop closure constraints are individually checked. If any one constraint's $\chi^2$ error is above a threshold, the individual constraint is rejected. A similar test is applied during an inter-cluster consistency check, where clusters are rejected from optimization if they are inconsistent with other clusters.

Sunderhauf and Protzel [38,39] developed a method that gives the back end data association authority referred to as switching constraints. This method adds an extra free variable, coined a switching variable, to each edge that could potentially be an outlier. The switching variable is restricted to be a value between 0 and 1. When a given switching variable takes a value of 0, there is no certainty associated with the particular edge, effectively removing it from the optimization. When the switching variable's value is 1, the edge maintains its original certainty. To prevent the

optimization from turning off all constrains, additional switch priors are added to each switching variable which induces a cost for every switching variable that is turned off.

In [40], Sunderhauf and Protzel compare max-mixture models, RRR, and switching constraints. They find that RRR outperforms both max-mixture and switchable constraints in real data sets, but underperforms in simulated datasets. In general, switchable constraints tend to outperfrom max-mixture with respect to both speed and accuracy.

Agarwal et al. [41] further develops the method of switching constraints. They explore the effect of a given switching variable on the optimization and are able to solve for the value of the switch variable explicitly. They call their method Dynamic Covariance Scaling (DCS). By using a closed form solution for switching variables, they are able to reduce graph complexity while maintaining robustness against outlying measurements. Another advantage of DCS is that it can be easily implemented in virtually any pose-graph SLAM algorithm.

Sunderhauf and Protzel extend switching variables to reject outlying GPS measurements in [42]. They use the pseudorange from each GPS satellite as an additional edge in their graph. A switching variable and associated switch prior are added to each of these edges. They are able to constrain error by giving 0 weight to edges identified as erroneous pseudoranges. This approach is advantageous in that it rejects individual satellite measurements, rather than the GPS position estimate. However, a GPS unit providing pseudorange measurements is required and it drastically increases the graph complexity because there can be upward of eight pseudorange measurements for each GPS position estimate received.

The remainder of this chapter is organized as follows: Section 4.1 outlines our approach to incorporate GPS in a robust back end to form a globally consistent map. We demonstrate our system in simulation (Section 4.2.1), hand-carried experiments (Section 4.2.2), and through flight tests on a multirotor aircraft (Section 4.2.3). To the best of our knowledge, we are the first to demonstrate a full system, featuring DCS to reject erroneous measurements, on a multirotor aircraft. Additionally, we are the first to simultaneously handle both loop closures and GPS in the back end.

## 4.1 Approach

The front end used to obtain the results in this chapter is the same front end as described in Chapter 3. Our navigation solution uses the RMEKF to fuse inertial and exteroceptive measurements and we construct a pose graph from odometry measurements provided by the front end. In this chapter, we extend the back end approach of Chapter 3 to include GPS measurements and to provide robustness to outliers.

### 4.1.1 GPS Integration

Due the nature of the relative navigation front end, it can be difficult to incorporate GPS as pseudo-relative measurements (e.g. differencing two GPS measurements). This is particularly true when GPS measurements are quite sparse, spanning multiple local coordinate frames. We opt to add GPS measurements directly to the back end using a virtual zero as in [35]. We use GPS position estimates rather than pseudorange measurements [42] from each satellite to reduce graph complexity and because common consumer GPS modules do not readily output pseudorange measurements.

The virtual zero is a fictitious node effectively representing the origin of an absolute coordinate system. In our implementation, we consider the virtual zero to be the origin of our local UTM zone. The virtual zero needs to be connected to the graph with an edge, which we call a virtual constraint. Though it can be connected to any node, we always associate it with the initial node in the graph. The virtual constraint is initially an arbitrary transformation with no certainty, in other words its associated information matrix (inverse of covariance matrix)is $\mathbf{0}$. Together, the virtual zero and virtual constraint allow the graph to translate and rotate about the absolute origin. Additional edges, introduced from GPS measurements, stretch or pull the virtual constraint until the graph is well aligned with the measurements. After incorporating several GPS measurements, the virtual constraint will approximate the global coordinate of its linked node. For this reason, the virtual constraint could be connected to the first node where a GPS measurement is received, and initialized as the GPS coordinate, though initializing it to the identity transform is adequate. Figure 4.1 visualizes the virtual zero and constraint with GPS measurements.

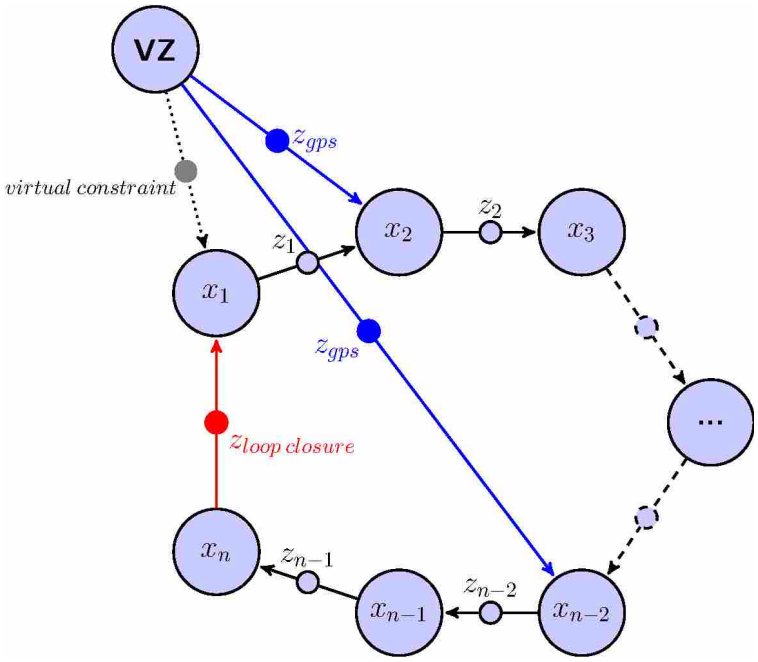



Figure 4.1: A virtual zero and virtual constraint are added to the graph to allow it to be translated and rotated freely on the global plane. Additional GPS measurements constrain the graph to align it with its true absolute position and orientation.

Several novel contributions were required to implement these ideas within the context of the relative navigation framework. In Section 4.1.2 we overcome problems introduced by using the virtual zero and constraint in the absence of GPS measurements. Section 4.1.3 discusses an approach to add intra-nodeframe GPS measurements. A method to avoid converging to local minima due to initial graph orientation is detailed in Section 4.1.4. Issues regarding the enormous scale of costs induced by GPS are remedied in Section 4.1.5. We conclude Section 4.1 by specifying the robust optimization algorithms used in Section 4.1.6 and run-time characteristics in Section 4.1.7.

### 4.1.2 Virtual Zero and Constraint in Absence of GPS

Together, the virtual zero and virtual constraint allow a graph to be translated and rotated on the global plane. This behavior is undesirable in the case that no GPS measurements have been received because the graph is free to shift unnecessarily. In other words, optimization is unconstrained when the graph incorporates a virtual constraint but does not add additional global constraints through GPS measurements. To mitigate this, before we have received GPS measurements, we make the virtual constraint the identity transform with infinite certainty, thus aligning

the graph with the vehicle's initial pose and globally constraining the graph. When the first GPS measurement is received, we can zero the confidence on the virtual constraint and allow GPS measurements to globally constrain the graph.

### 4.1.3 Intra-nodeframe GPS Measurements

Because the front end itself only outputs odometry measurements between nodeframes, when a GPS measurement is received an additional node must be defined at the vehicle pose where the GPS measurement was received. Two additional edges are added to the pose graph:

1. The transformation and covariance from the last nodeframe to the current state is used as an odometry edge to define the new node where the GPS measurement was received.

2. An edge from the global coordinate system (virtual zero) to the current state. This transformation is defined by the incoming GPS measurement, which in our implementation is transformed into the local UTM zone, and its associated covariance.

This method allows us to apply multiple GPS measurements as constraints with respect to a single locally established coordinate frame, as shown in Figure 4.2. We only use the north and east GPS measurements, neglecting altitude and heading.

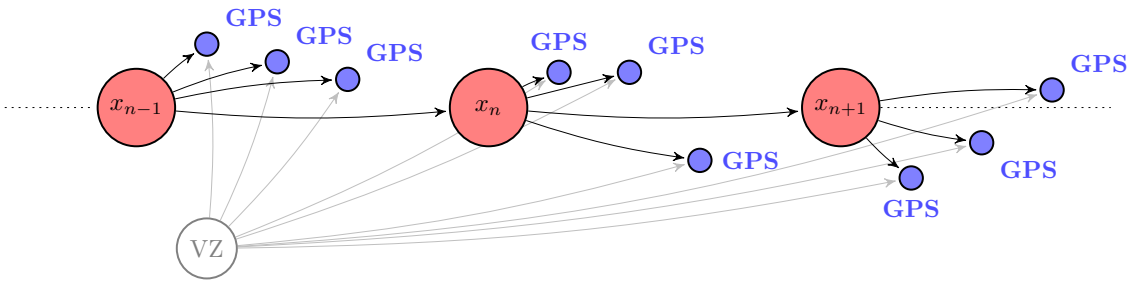



Figure 4.2: Additional intra-nodeframe odometry edges are established that correspond to the physical location where the GPS measurement was received. The GPS measurement is added as an edge between the virtual zero and the new node established by inter-nodeframe odometry.

### 4.1.4 Informed Initial Guess

Including the virtual zero and constraint in our graph allows the map to translate and rotate on the absolute plane to align with GPS measurements, but we find that in certain situations the initial guess of node positions is not accurate enough to result in the global minimum solution. The scenario may occur when the true vehicle trajectory is opposite the believed vehicle trajectory. In other words, because our initial vehicle's orientation is unknown, we assume that the forward direction is north on the absolute coordinate system, whereas the true initial forward direction may be south.

We implement Algorithm 1 that uses the initial $\chi^2$ error of the graph to intelligently adjust the vehicle's initial orientation. The algorithm assumes that at least two GPS measurements have been received prior to optimization, otherwise the initial orientation is irrelevant. After graph construction, we calculate its initial $\chi^2$ error. We then iteratively rotate the graph nodes by an angle $\theta$ and recompute the initial $\chi^2$ error. The graph orientation that results in the minimum $\chi^2$ error is selected for future optimization. Results of this algorithm are presented in Section 4.2.2.

**Algorithm 1** Initial Guess algorithm

```
1:  procedure INITIALGUESS(x)
2:      Initialize vector of angles θ, length n
3:      Initialize empty vector e, length n
4:      for i = 1 : n do
5:          x_i ← x rotated by θ_i
6:          e_i ← χ²(x_i)
7:      end for
8:      min ← index of min(e)
9:      x_min ← x rotated by θ_min
10:     return x_min
11: end procedure
```

### 4.1.5 Scale of GPS Induced Costs

During graph construction, we assume the map is being built near the origin of a UTM coordinate frame; however, as we receive GPS measurements, we find that we may be far from

it. When there are significant costs induced from GPS updates, on the order of hundreds of thousands of meters, round off errors in computing the Hessian matrix **H** (Appendix A), during Gauss-Newton optimization, become significant. The round-off errors cause the Hessian to become non-positive-semi-definite, preventing g2o from completing the optimization. We suggest two methods to overcome this problem.

1. Subtract the first GPS measurement from all subsequent incoming GPS measurements. This effectively translates the GPS measurements to the UTM origin local area.

2. Add the first GPS measurement to all previous and future vehicle poses. This translates the graph nodes to the general area of the GPS measurements.

Both of these methods reduce the scale of GPS-induced errors and subsequent round-off errors. Using these methods, we are able to successfully optimize graphs that would otherwise fail because of the Hessian's definiteness. We typically use the former method because we generally assume fewer GPS measurements than nodes in the graph, therefore we need only to adjust a few GPS measurements rather than potentially thousands of nodes in the graph.

### 4.1.6 Robust Optimization

We use the popular g2o library for graph construction and optimization in our experimental hand-carried and flight tests. In this paper Gauss-Newton optimization is used to be consistent with the MATLAB simulation, but other algorithms offered in the g2o library could easily be substituted, such as Levenberg-Marquardt. Additionally, we employ DCS on edges that could be outliers, specifically loop closures and GPS.

### 4.1.7 Run Time

Because our front-end flight-critical processes are completely decoupled from the back end, it is irrelevant whether the back-end processes are run on or off board. The results in this thesis were processed off board to facilitate analysis; however, we have demonstrated in previous experiments that we can run real-time on-board without causing any degradation to the front end.

Additionally, the back end can be configured to optimize the graph at every informing measurement (i.e. GPS measurement or loop closure), at opportunistic intervals, or as dictated by the front end. We perform batch optimization for consistency. It is noted that incremental optimization can result in differing local minima solutions than doing batch optimization.

For small-scale graphs, such as the those presented in Sections 3.5 and 4.2, completing ten iterations of Gauss-Newton batch optimization takes less than 0.22 seconds on the i7 processor. We refer the reader to [41] for more details on DCS runtime performance for large-scale graphs.

## 4.2 Results

The following sections present results that validate our approach to generate globally consistent maps in GPS-degraded environments. Simulation results highlight that we are able to maintain globally consistent maps in the presence of both GPS outliers and false-positive loop closures. The hand-carried results highlight the need for methods to ensure global convergence when incorporating intermittent and erroneous GPS measurements. Finally, flight test results demonstrate our ability to construct globally consistent and aligned maps in practice.

### 4.2.1 Simulation

Using MATLAB, we synthetically create edges from a 2D robot trajectory that would traditionally be provided from the SLAM front end. The edges are corrupted by zero-mean Gaussian noise to induce drift throughout the robot trajectory. The back end compounds the generated edges to initialize a starting guess of global node positions as shown in Figure 4.3.

Loop closures are detected using the uncorrupted robot trajectory. A loop closure edge is added between non-consecutive nodes when the Euclidean distance between the true node positions is less than a quarter meter. We calculate the relative transformation between the two nodes and corrupt it with zero-mean Gaussian noise. The transformation and associated covariance are added to the graph as an edge.

We establish a virtual zero node as the first node in the graph. The information matrix associated with the virtual constraint is $\mathbf{0}$. GPS measurements are generated randomly throughout the robot trajectory. The GPS measurement is synthesized by corrupting true global position of the

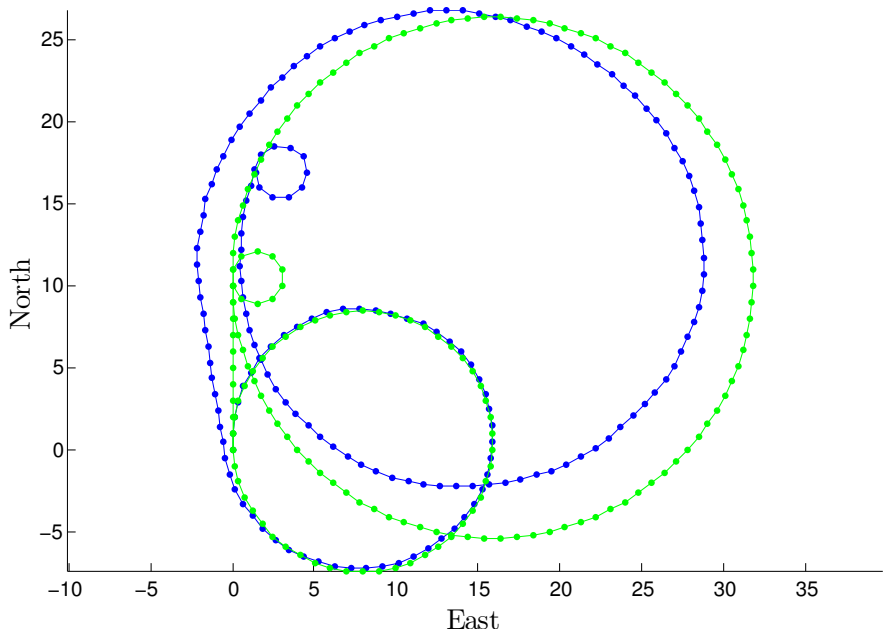



Figure 4.3: An example path generated using the MATLAB back-end simulator. The corrupted vehicle path (blue) drifts from truth (green).

node with white noise. An edge, representing the GPS measurement, is added to the graph between the virtual zero and the node where the measurement was received. This method is analogous to subtracting the first GPS measurement from all subsequent GPS measurements in a UTC coordinate frame, as described in Section 4.1. This reduces the scale of errors between the estimate node positions and their absolute positions, forming a positive-semi-definite Hessian matrix to enable optimization.

DCS is implemented by calculating the switching variable, $s_{ij}$, for loop closure and GPS edges, using the closed formed solution

$$s_{ij} = \min\left(1, \frac{1}{1+\chi_{ij}}\right) \tag{4.1}$$

where $\chi_{ij}$ is the cost from the original constraint,

$$\chi_{ij} = (\mathbf{h}_{ij}(\mathbf{x}) - \mathbf{z}_{ij})^T \mathbf{\Omega}_{ij} (\mathbf{h}_{ij}(\mathbf{x}) - \mathbf{z}_{ij}).$$

Here $\mathbf{h}_{ij}(\mathbf{x})$ is the measurement function that evaluates the expected measurement between the two nodes $i$ and $j$, and $\mathbf{z}_{ij}$ is the actual measurement. After computing $s_{ij}$ for each edge that could be

an outlier, we scale the information matrix, $\mathbf{\Omega}_{ij}$, of the corresponding constraint by $s_{ij}^2$. Finally, we formulate the following least-squares problem to find the most likely arrangement of nodes given the edges, an extension of Equation A.1 (see Appendix A).

$$\begin{aligned}\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \sum_{ij} & \underbrace{(\mathbf{h}_{ij}(\mathbf{x}) - \mathbf{z}_{ij})^T \mathbf{\Omega}_{ij}^{\mathrm{od}} (\mathbf{h}_{ij}(\mathbf{x}) - \mathbf{z}_{ij})}_{\text{Odometry Constraints}} \\ & + \sum_{ij} \underbrace{(\mathbf{h}_{ij}(\mathbf{x}) - \mathbf{z}_{ij})^T s_{ij}^2 \mathbf{\Omega}_{ij}^{\mathrm{lc}} (\mathbf{h}_{ij}(\mathbf{x}) - \mathbf{z}_{ij})}_{\text{Loop Closure Constraints}} \\ & + \sum_{ij} \underbrace{(\mathbf{h}_{ij}(\mathbf{x}) - \mathbf{z}_{ij})^T s_{ij}^2 \mathbf{\Omega}_{ij}^{\mathrm{gps}} (\mathbf{h}_{ij}(\mathbf{x}) - \mathbf{z}_{ij})}_{\text{GPS Constraints}}\end{aligned} \tag{4.2}$$

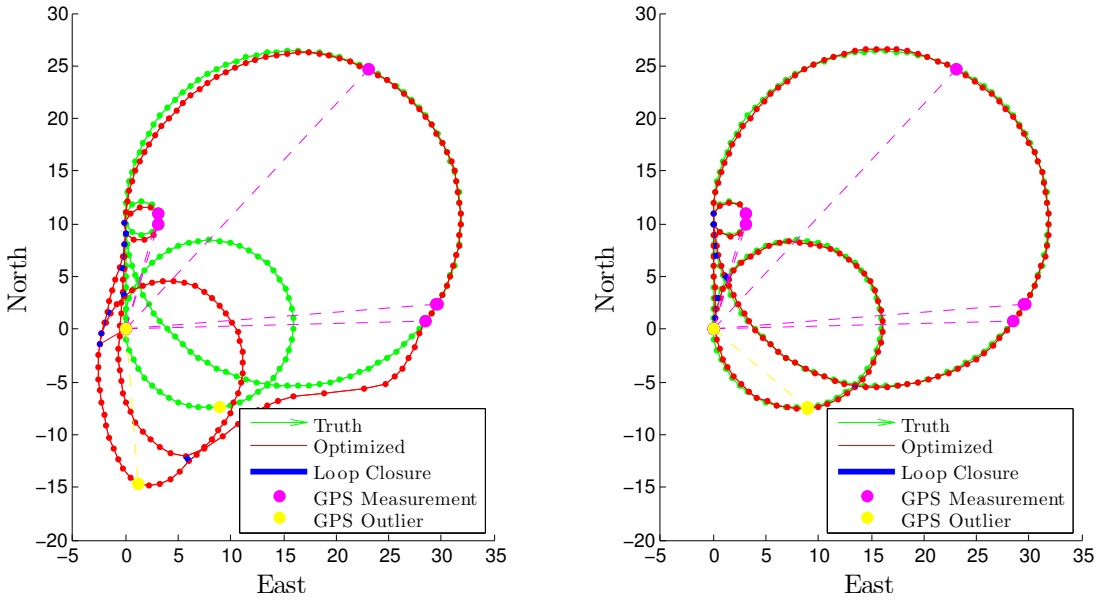
We use Gauss-Newton techniques to iterate toward $\mathbf{x}^*$. Taking the derivative of the first-order Taylor approximation of (4.2) results in the linear system

$$\mathbf{H}\Delta\mathbf{x}^* = \mathbf{b}. \tag{4.3}$$

Equation (4.3) can be solved to find the steps $\Delta\mathbf{x}^*$ that lead to an optimal node arrangement. We efficiently formulate the Hessian $\mathbf{H}$ and the vector $\mathbf{b}$ by taking advantage of their sparse structure [9]. Ten iterations of Gauss-Newton optimization are run after the entire graph has been constructed rather than incrementally throughout the robot trajectory.

Figure 4.4 shows a synthetic robot trajectory interspersed with a total of 6 GPS measurements, one of which is an outlier. The graph is optimized using non-robust optimization (left) as well as using DCS (right). In this case, the non-robust optimizer does a relatively good job of translating and rotating the map into is proper absolute position, however it is clear that the single outlier has severely degraded the map quality. DCS successfully detects the erroneous GPS measurement and essentially rejects it from the solution, significantly reducing the error of the robustly optimized graph. This result is representative of numerous realizations of similar simulations.

The values of $s_{ij}$ for each edge are shown in Figure 4.5. DCS was able to give the erroneous GPS measurement a weight of 0 while maintaining a weight of 1 for all valid GPS measurements. Because Equation 4.1 will never be 0, unless $\chi_{ij}$ is infinite, we say $s_{ij}$ is 0 if it is small enough to reject the edge. As we add substantially more valid GPS measurements, DCS begins to decrease




(a) Traditional least-squares

(b) DCS

Figure 4.4: Graph generated using the MATLAB simulator described in Section 2.5. A single outlier in traditional least-squares optimization (left) significantly distorts the graph. Robust least-squares methods, such as DCS, can detect and reject outliers, resulting in more accurate global maps (right).

the weight of some of these measurements. This is expected because all of the measurements are corrupted with white noise, DCS chooses the best representative set of measurements and begins to devalue measurements that are not as well aligned.

In Figure 4.6, we add false-positive loop closures in addition to outlying GPS measurements. DCS maintains a globally consistent map even in the presence of both types of gross errors. We show the histogram of switching variable values at the end of optimization in Figure 4.7. Again we see that no valid GPS measurements are turned off, nor are any legitimate loop closures, but all of both types of outlying measurements are given a weight of 0.

Although this approach can successfully optimize small scale graphs in the presence of GPS outliers, it is currently susceptible to problems involving biased measurements. This problem is discussed with preliminary treatment in the following section.

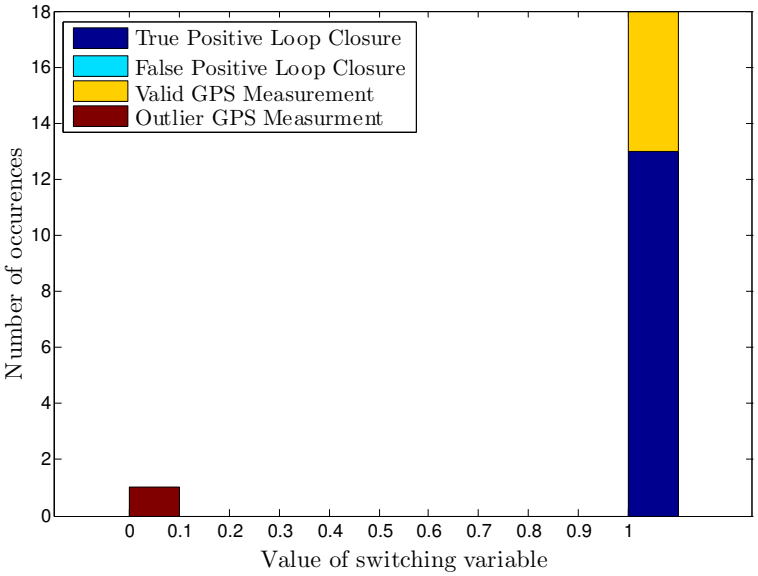
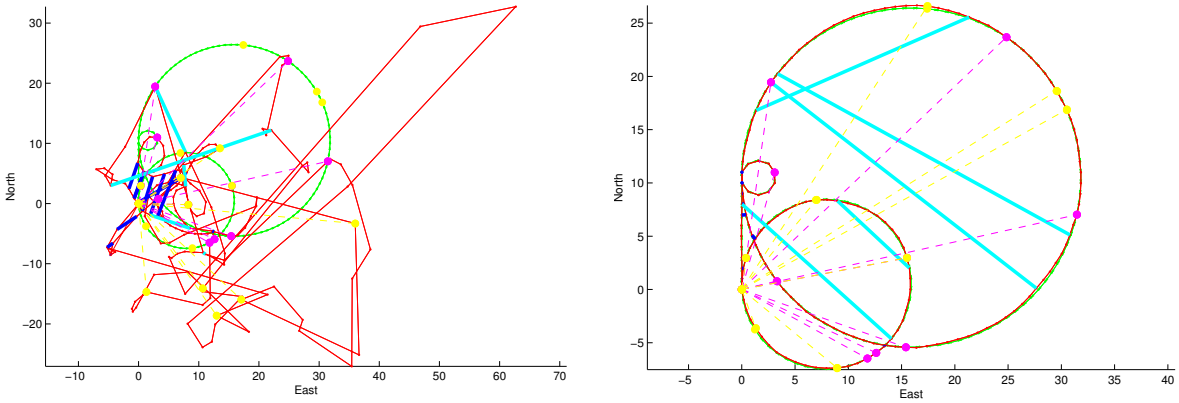



Figure 4.5: Histogram displaying the switching variables values associated with the optimized graph in Figure 4.4. All valid GPS measurements and loop closures are unaffected by DCS as indicated by the switching variables having value 1. The single oulier's switching variable (red) takes a value of 0, effectively rejecting the edge from optimization.




(a) Traditional least-squares

(b) DCS

Figure 4.6: The severely distorted graph (left) is the result of traditional least-squares operating on both false-positive loop closures and outying GPS measurements. DCS (right) is able to recognize and reject both types of outliers.

### 4.2.2 Hand Carried

The following hand-carried results highlight (1) the need for an algorithm to refine the initial orientation of the pose graph, such as Using Algorithm 1 and (2) issues that arise when the first GPS measurement is an outlier or biased.

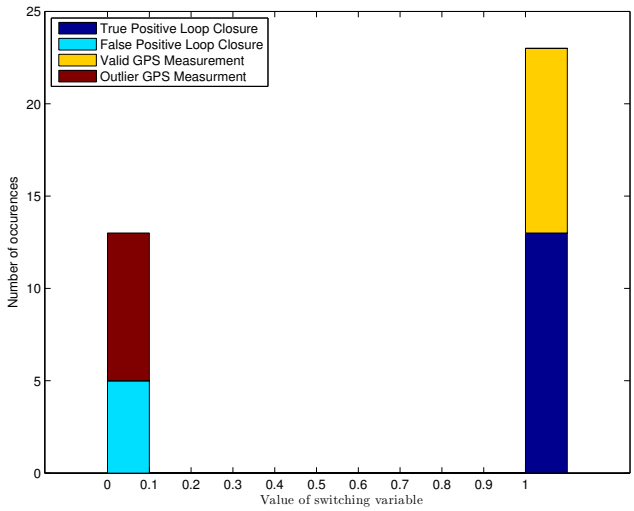



Figure 4.7: Values of switching variables associated with the graph in Figure 4.6. All false-positive loop closures and outlying GPS edges were detected and assigned a value of 0, while all valid measurements retained a value of 1.

### Initial Orientation Refinement

The following hand-carried results highlight the need for an algorithm to refine the initial orientation of the pose graph, such as using Algorithm 1. An RGB-D camera with the DEMO algorithm (Section 3.2.2) and the Ublox LEA-6T GPS module were carried around the inside of the baseball park shown in Figure 4.8. The collected data are used to construct the graphs in Figure 4.9, where GPS measurements were downsampled to simulate intermittent GPS. Figure 4.9a shows the original graph orientation resulting from the experiment. The believed trajectory (blue) is nearly 180° offset from the "true" GPS trajectory. Figure 4.9b is the result of optimizing the original graph without refining its initial pose. The optimizer converges to a local minima that does not accurately represent the true path. When Algorithm 1 is used, the graph is rotated 165° before optimization. The optimized result, shown in Figure 4.9c, is clearly a much more accurate representation of the path. This method was successful at aligning the graph with its true global position and orientation with as few as two GPS measurements, although odometric drift was substantially more noticeable.

Figure 4.10 shows the initial and final $\chi^2$ error versus initial heading angles. From the plots, we find there is only a 65° band where the optimization converges to a local minima. Since optimization is somewhat insensitive to the initial orientation (Figure 4.10b), it may be sufficient to

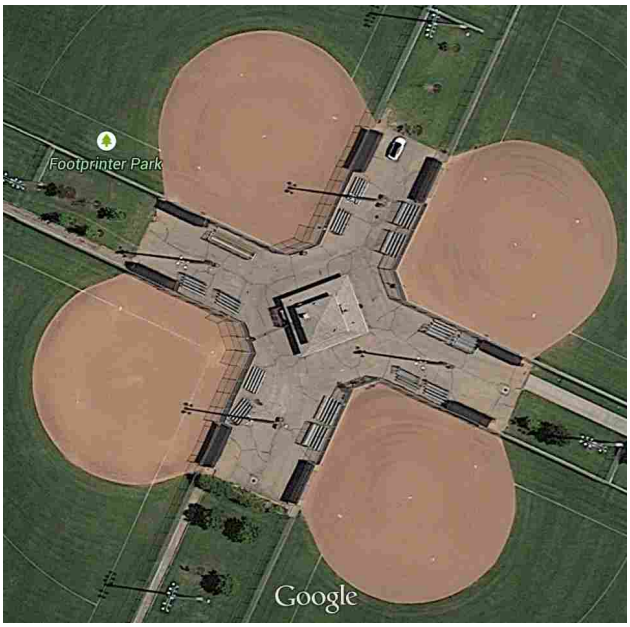
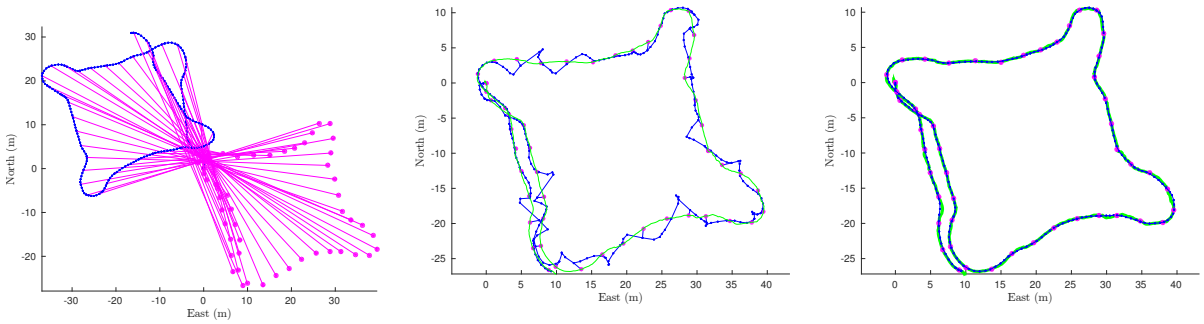



Figure 4.8: An image of Footprinter Park, the baseball park where hand-carried and flight experiments were performed, taken from Google Maps. The experimental paths follow the inside of the fence, turning before reaching each light post.




(a) Original graph orientation

(b) Optimization before refined initial orientation

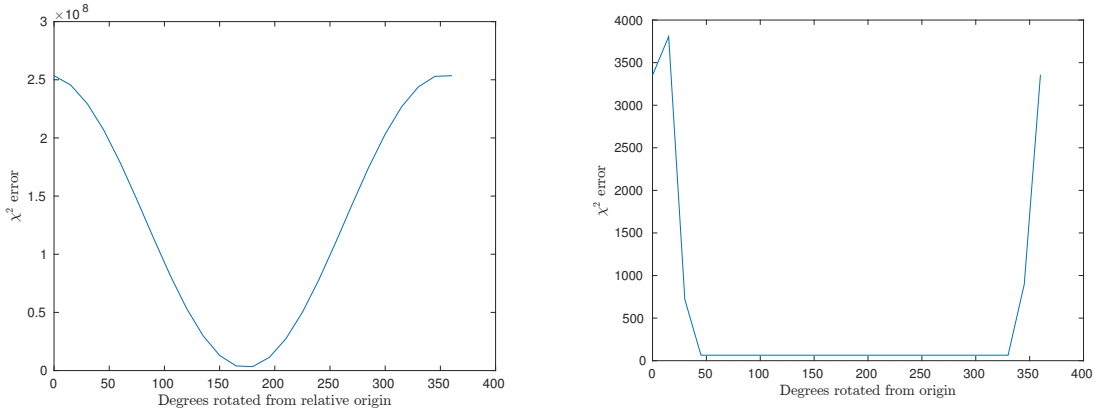
(c) Optimization after refined initial orientation

Figure 4.9: A hand-carried dataset that shows the importance of refining the initial orientation of the graph. When the original graph (left) is optimized without modifying its orientation, the solution converges to a local minima (that is not consistent with the actual trajectory. After refining the initial graph orientation, using Algorithm 1, and optimizing, the resulting graph fits closely with GPS truth (green).

evaluate the initial $\chi^2$ error at only a few angles, e.g. the original orientation and rotated by 180°, to ensure that the orientation is relatively close to truth.

These results were obtained from batch optimization, but it is worth noting that this algorithm needs to be run only once even when performing incremental optimization. The algorithm

can be run, after sufficient global information has been received, to find a good guess of the vehicle's starting pose. The same pose can be used while performing incremental optimization and only needs to be rerun if confidence in the guess is lost.




(a) Initial $\chi^2$ error

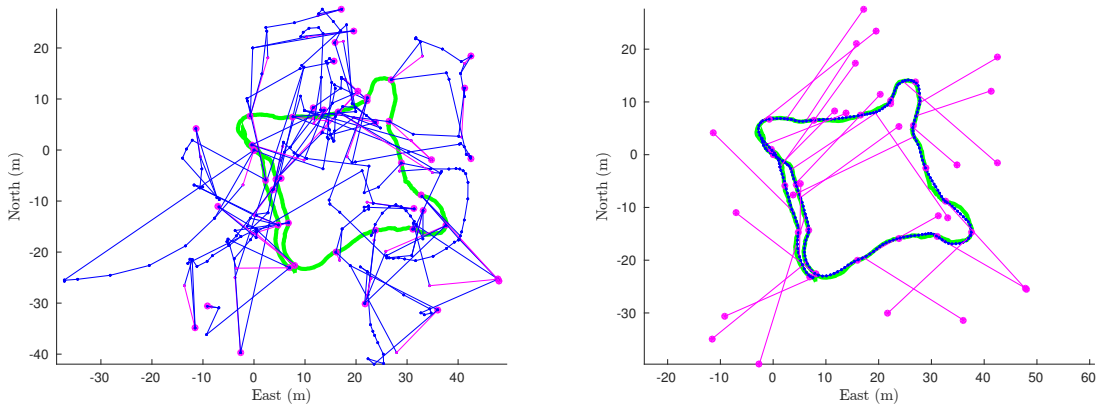
(b) Resulting $\chi^2$ error after 10 iterations

Figure 4.10: The initial $\chi^2$ error after rotating the original graph through 360° (left) shows that the $\chi^2$ reduces substantially as the graph becomes closely aligned with its true orientation. The $\chi^2$ error after 10 iterations of optimization over the same rotations (right) indicates that optimization is relatively insensitive to initial orientation, but that there is a region that results in suboptimal, local minima.

**First GPS Measurement is Erroneous**

Figure 4.11 shows that are approach is capable of maintaining globally consistent maps in the presence of numerous GPS outliers. However, inaccurate map reconstruction can occur if the first GPS measurement received by the back end is either biased or an outlier. This scenario is quite likely in practice. For example, if a UAS begins its mission indoors, it will acquire GPS for the first time when it exits the building. During the exit, the UAS will be prone to receiving GPS measurements corrupted by multipath error since multipath error is common when operating in close proximity to buildings.

This is problematic in our approach where we subtract the first GPS measurement from all subsequent measurements to reduce to the scale of costs induced by GPS, as described in Section 4.1.5. If the first GPS measurement is erroneous, this method naively reduces the cost of the erroneous measurement. In Figure 4.12a, we show an optimized graph where the first four GPS




(a) Traditional Least-squares (b) DCS

Figure 4.11: Comparison of graphs using traditional least-squares (left) and DCS (right) in the presence of numerous outlying GPS measurements. Using robust least-squares optimization and other methods described in this chapter, we are able to make an accurate reconstruction of the vehicle trajectory.

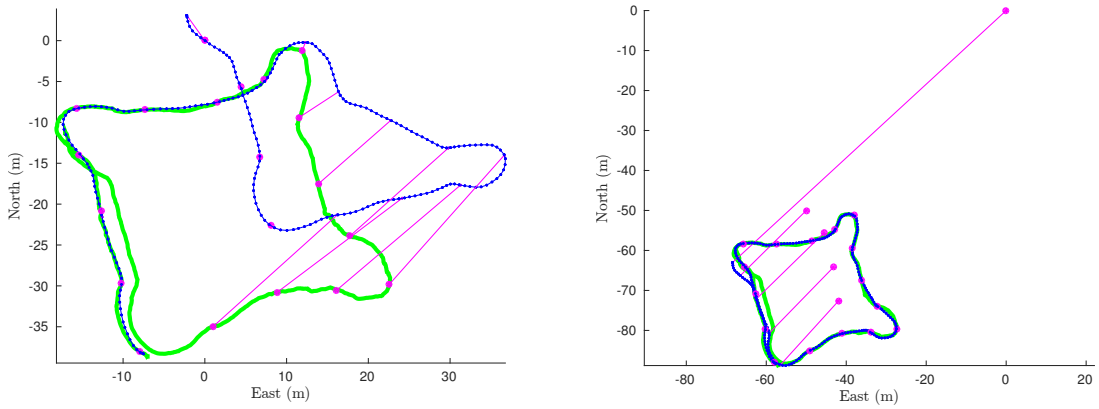
measurements are biased. We see that DCS does not reject the biased measurements, but rather rejects the subsequent valid measurements, and results in a distorted graph.

Preliminary experiments show that if we perturb the GPS measurements by a small amount, in our case 50 meters in the north and east directions, DCS rejects the biased measurements but results in an offset graph (Figure 4.12b). After optimization this can easily be compensated for by negating the induced offset from each graph node. The same behavior is consistent when the first GPS measurement is an outlier.

Alternative methods for ensuring that the first GPS measurements do not derail the solution should be explored. We expect that reducing the certainty of the first GPS measurements will result is globally consistent maps but leave it as an area of future work.

### 4.2.3 Flight Tests

Two flight tests were performed to validate the back end. First, the indoor flight from Section 3.5 is used to show that the system is robust to false-positive loop closures. Secondly, an outdoor flight test is used to show intermittent and degraded GPS integration. We separate the cases of false-positive loop closures and outlying GPS measurements. Due to the camera




(a) Before GPS Perturbation

(b) After GPS Perturbation

Figure 4.12: Optimization results before (left) and after (right) perturbing GPS measurements. Perturbing the GPS measurements results in a more globally consistent, but offset graph.

limitations presented in Section 3.2.2, we are unable to compute loop-closure transforms outdoors during a flight tests, and as a result we have not experimentally validated the cases simultaneously.

### Indoor Flight Test

There were not any registered false-positive loop closures during the flight described in Chapter 3. In post process, we inject a synthetic false-positive loop closure between two random nodes. The resulting graph is shown in Figure 4.13. When the graph is optimized without using a robust least-squares algorithm, it becomes severely distorted (center), rendering it useless for high-level path planning. However, we see that when DCS is employed on loop-closure edges, the false positive is detected and rejected, yielding a globally consistent map (right).

### Outdoor Flight Test

Flight tests were conducted to show the efficacy of our approach on real hardware and with actual GPS measurements. The multirotor platform, equipped with the sensors described in Section 2.1, was flown around the perimeter of a building by providing commanded body-frame velocities in the forward and right directions. The 330 meter true flight path (green) and experimentally collected GPS path (yellow) are shown in Figure 4.14. Because of the sensor limitations described in Section 3.2.2, the flight path is constrained to be close to the building's walls where depth

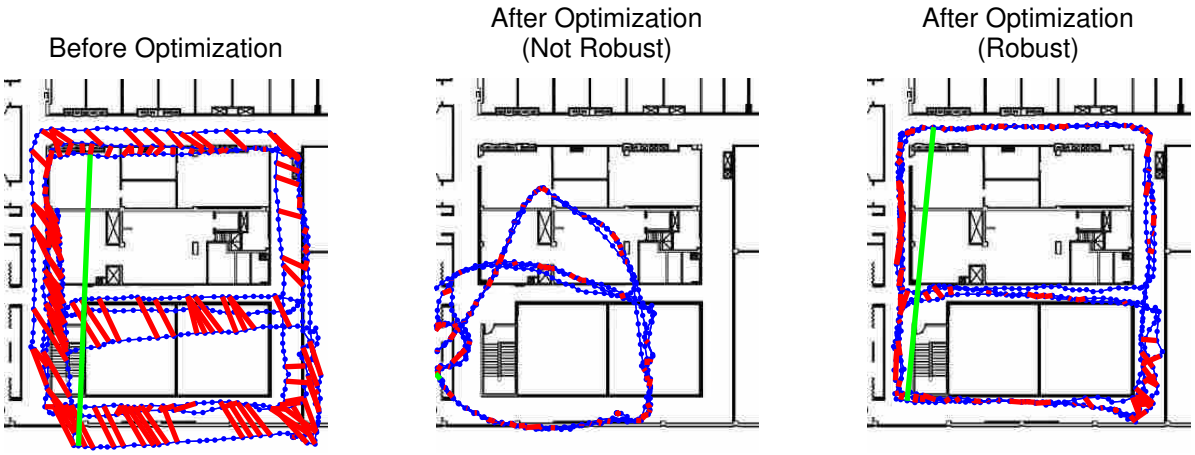



Figure 4.13: A false-positive loop closure (green) is injected in the graph formed from the indoor flight test from Chapter 3 (left). When traditional least-squares optimization is used, the resulting graph (center) is severely deformed; however, when DCS is employed the optimized result (right) is a quite accurate representation of the true map.

information is available. However, flying in close proximity to the building causes degradation to GPS measurements. We can see both outliers and biased measurements in the resulting GPS path. GPS measurements were downsampled to simulate intermittent GPS in the following results.

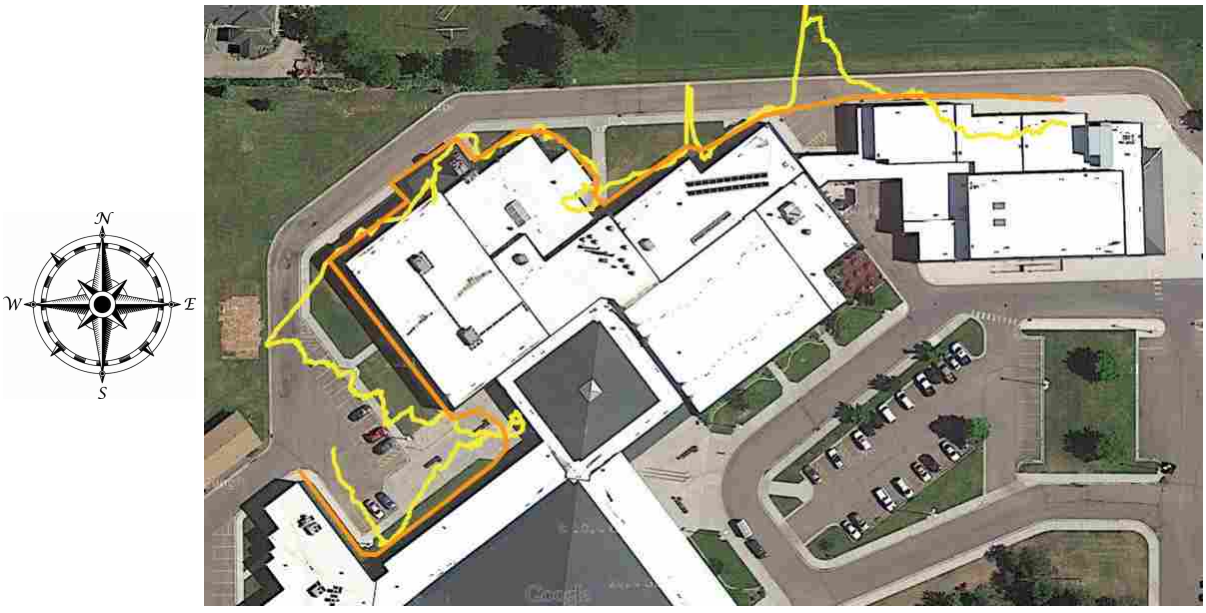



Figure 4.14: Experimental flight path (orange) and corresponding GPS path (yellow) shown on a map of the building where outdoor flight tests were conducted. The path begins on the west side of the building. It is clear that much of the GPS path is degraded.

The RMEKF described in Section 3.2.1 fused visual odometry, IMU, and altimeter measurements to estimate states and provide odometry measurements to the global back end for graph construction. Figure 4.15 shows the resulting unoptimized graph after refining the initial orientation using Algorithm 1. The magenta dots indicate the GPS measurement that was received at its connected node to illustrate the sparsity of GPS measurements.

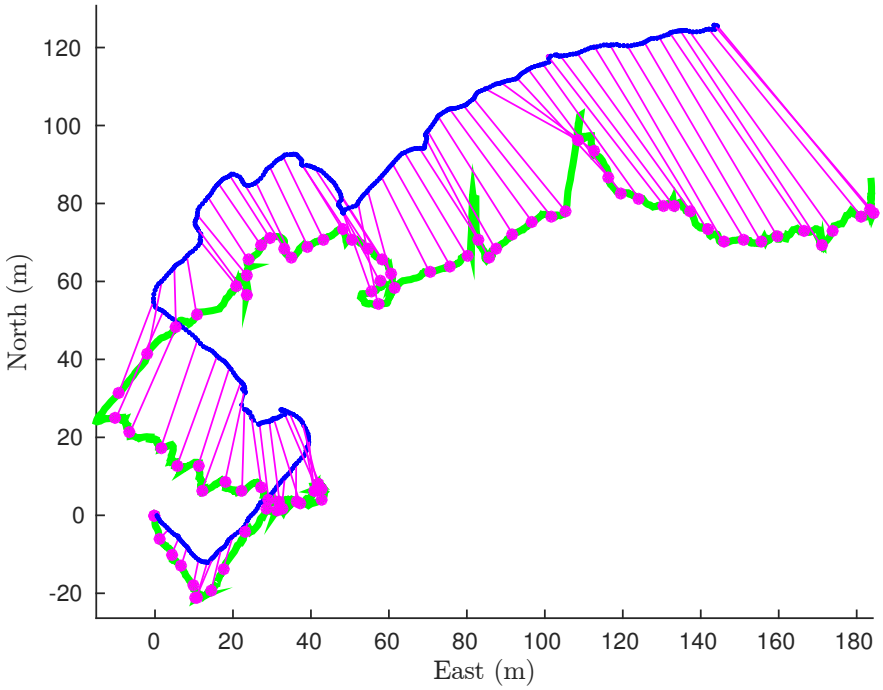



Figure 4.15: The graph constructed by traversing the perimeter of the building. Odometry nodes and edges are shown in blue. The magenta dots and lines represent the GPS measurement received at the connected node. Green is the complete GPS path.

Figure 4.16 shows that without robust optimization (red), every GPS measurement is used to optimize the graph and results in a map that is distorted from the true flight path. By using DCS (blue) on GPS edges, we are able to mitigate GPS errors and much more accurately reconstruct the global vehicle path.

We note the inconsistency at the beginning (west side) of the robustly optimized map. As discussed in Section 4.2.2, when the initial GPS measurements are erroneous they are not easily rejected by our approach. This is a particularly challenging dataset because the majority of the GPS

Figure 4.16: Comparison of refined maps using traditional and DCS least-squares optimization. The result of traditional least-squares (red) is clearly distorted as it uses all GPS measurements for optimization. The map refined using DCS (blue) rejects erroneous GPS measurements and results in a more consistent and globally aligned map. The true experimental path is displayed in green.

measurements are degraded. In this instance, even perturbing the graph does not fully compensate for the initial bias in GPS measurements.

If we can detect that GPS measurements are inaccurate before optimization, for example from the GPS receiver's reported uncertainty, then we can omit them from optimization. Figure 4.17 demonstrates this by rejecting all GPS measurements in the red-shaded area. The result is more consistent with the true flight path and the error in the red region is attributed to drift in odometry rather than GPS. Characterizing GPS uncertainty and developing methods to reject initial erroneous measurements is an important topic of future research.

To better illustrate our ability to incorporate non-uniformly intermittent GPS measurements, we artificially create GPS-denied regions in the flight path. Figure 4.18 shows these regions and the resulting optimized path. Even with large GPS-denied areas, the resulting map is nearly identical to the graphs that use significantly more GPS measurements.

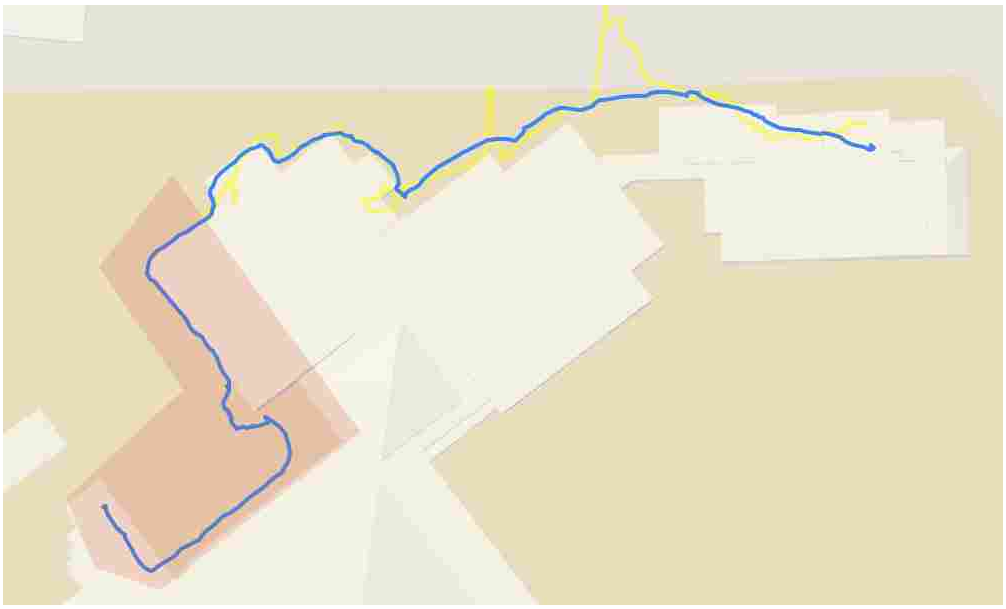


Figure 4.17: Demonstration of rejecting initial erroneous GPS measurements to compensate for initial bias. The red-shaded region corresponds to the area that GPS measurements are rejected.

## 4.3 Conclusion

The approach taken in this thesis is capable of integrating sparse GPS measurements, as well as detecting and rejecting outliers, to maintain a globally consistent and aligned map. Using GPS position estimates, rather than pseudoranges, is sufficient to constrain the map and significantly reduces graph complexity. A drawback of using position estimates is that we reject an entire set of pseudorange measurements rather than rejecting a single pseudorange and utilizing information from the remaining valid measurements. In other words, we waste potentially useful information to ensure that erroneous information is rejected. Alternative methods of gleaning valid information from erroneous position estimates is an area of future work.

Our approach to refining the initial orientation of the graph prior to map optimization helps to ensure that the graph will not converge to a local minima. This work validates the relative navigation approach as a solution to intermittent and degraded GPS navigation despite not using any global information in the front end.

As a matter of future work, we seek to close the loop between the global high-level and relative low-level path planners. This would allow missions that utilize the globally consistent map, further validating the contributions of this research. For example, the vehicle could be commanded

Figure 4.18: Maintaining globally consistent and aligned maps in the presence of non-uniformly intermittent GPS measurements is demonstrated by artificially creating GPS-denied regions. The red-shaded polygons indicate the GPS-denied areas. The optimized path is displayed in blue.

to a desired GPS coordinate, even within a building, or commanded to return to a previously visited location along the shortest known path.

# CHAPTER 5. CONCLUSIONS AND RECOMMENDATIONS

This chapter offers concluding remarks about the work presented and suggests future directions that could be taken to further the research in this area.

## 5.1 Conclusion

We have shown through validating experimental flight tests that the relative navigation framework is a viable solution to autonomous navigation is GPS-degraded environments. In particularly, the indoor results in Section 3.5 show that we can accurately reconstruct the vehicle's global states without any negative effects on relative state estimation and control.

In Section 4.2, we showed simulation, hand-carried, and flight test results that successfully incorporate intermittent GPS measurements as direct measurements to the back end, by using a virtual zero and constraint. Although the virtual zero and constraint allow for GPS measurements to be added as edges in the pose graph, additional methods are required to ensure that the graph is globally constrained in the absence of GPS, add intra-nodeframe GPS measurements, refine the initial orientation of the graph, and ensure that the scale of GPS induced costs allow for successful optimization. The integration of these methods allows our unique back end to globally align graphs with as few as two GPS measurements and maintains globally consistent maps in the presence of numerous outliers. From our results, we can conclude that using consumer grade GPS modules to provide position estimates, rather than pseudorange measurements from each satellite, is sufficient to identify and reject GPS errors.

## 5.2 Future Work

The following is a list of research directions that could be taken to further extend this work. Suggestions include possible avenues of work on both the relative front end and global back end.

- Investigate methods of closing the loop between the back end and the front end by means of a high-level path planner. This would show the feasibility of using a map made by the vehicle to achieve high-level goals, for example navigating to a GPS coordinate or returning to a previously visited location along the graph.

- Throughout flight tests, we experienced a variety of visual odometry errors.

    1. When only plain walls are in the camera's field of view, the features that are detected (if any), are coplanar. This makes it impossible to accurately calculate the camera's translation. Research methods to opportunistically gimbal the camera toward areas that are feature rich.

    2. Features that are reflections from reflective surfaces (e.g. windows, polished linoleum and wood floors) do not behave the same as non-reflected features. When VO algorithms track reflected features egomotion estimates are less accurate. Explore avenues to make VO robust to reflected features.

    3. Due to sensor limitations mentioned in Section 3.2.2, the current RGB-D sensor utilized on our platform is not an ideal choice for outdoor navigation. Integrate different sensors, such as laser scanners, to supplement VO outdoors.

- In our approach, if the first received GPS measurement is erroneous then the optimization results in a distorted graph. Although we have shown preliminary results that solve this issue by perturbing all GPS measurements (Section 4.2.2), more robust methods could be developed. For example, reducing the certainty of the first few GPS measurements.

- We use GPS position estimates rather than pseudorange measurements because of our focus on using consumer-grade GPS modules. This leads to wasting valid pseudorange information to reject outlying position estimates (Section 4.3). DCS, rather than switching constraints, could be used with pseudoranges to salvage information from valid measurements while still reducing graph complexity. Additional methods of gleaning valid information from erroneous position estimates can be researched.

## REFERENCES


[1] M. A. Goodrich, B. S. Morse, D. Gerhardt, J. L. Cooper, M. Quigley, J. A. Adams, and C. Humphrey, "Supporting Wilderness Search and Rescue using a Camera-Equipped Mini UAV," *J. F. Robot.*, vol. 25, no. 1-2, pp. 89–110, 2008. 1

[2] S. Herwitz, L. Johnson, J. Arversen, R. Higgins, and J. Leung, "Precision Agriculture as a Commericial Application for Solar-Powered Unmanned Aerial Vehicles," in *AIAA 1st Tech. Conf. Work. Unmanned Aerosp. Veh.*, 2002. 1

[3] N. Schwartz, "The United States as an Aerospace Nation: Challenges and Opportunities," in *IFPA Fletcher Conf. Natl. Secur. Strateg. Policy*, Medford, MA, 2010. 1

[4] R. C. Leishman, T. W. McLain, and R. W. Beard, "Relative Navigation Approach for Vision-Based Aerial GPS-Denied Navigation," *J. Intell. Robot. Syst. Theory Appl.*, vol. 74, no. 1-2, pp. 97–111, 2014. 2, 16, 21, 28, 30

[5] H. Surmann and R. Worst, "New Applications with Lightweight 3D-Sensors," p. 171, 2006. 3

[6] H. Durrant-Whyte and T. Bailey, "Simultaneous Localization and Mapping: Part I," *IEEE Robot. Autom. Mag.*, vol. 13, no. 2, pp. 99–108, Jun. 2006. 4

[7] J. C. MacDonald, "Efficient Estimation for Autonomous Multi-Rotor Helicopters Operating in Unknown, Indoor Environments," Ph.D. dissertation, Brigham Young University, 2012. 4

[8] F. Lu and E. Milios, "Globally Consistent Range Scan Alignment for Environment Mapping," *Auton. Robots*, vol. 4, no. 4, pp. 333–349, Oct. 1997. 4, 26

[9] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A Tutorial on Graph-Based SLAM," *IEEE Intell. Transp. Syst. Mag.*, vol. 2, no. 4, pp. 31–43, 2010. 4, 26, 48, 68

[10] A. Glover, W. Maddern, M. Warren, S. Reid, M. Milford, and G. Wyeth, "OpenFABMAP: An Open Source Toolbox for Appearance-Based Loop Closure Detection," in *IEEE Int. Conf. Robot. Autom.*, 2012, pp. 4730–4735. 4, 26

[11] N. Sunderhauf and P. Protzel, "BRIEF-Gist - Closing the Loop by Simple Means," in *IEEE Int. Conf. Intell. Robot. Syst.*, Sep. 2011, pp. 1234–1241. 4

[12] C. Stachniss, U. Frese, and G. Grisetti, "OpenSLAM." [Online]. Available: http://www.openslam.org 5, 29

[13] S. Grzonka, G. Grisetti, and W. Burgard, "A Fully Autonomous Indoor Quadrotor," *IEEE Trans. Robot.*, vol. 28, no. 1, pp. 90–100, Feb. 2012. 5, 17, 37

[14] G. Grisetti, C. Stachniss, and W. Burgard, "Nonlinear Constraint Network Optimization for Efficient Map Learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 10, no. 3, pp. 428–439, 2009. 5, 29

[15] A. Bachrach, A. De Winter, R. He, G. Hemann, S. Prentice, and N. Roy, "RANGE - Robust Autonomous Navigation in GPS-Denied Environments," in *IEEE Int. Conf. Robot. Autom.*, vol. 28, no. 5, Sep. 2010, pp. 1096–1097. 5, 17, 37

[16] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Incremental Smoothing and Mapping," *IEEE Trans. Robot.*, vol. 24, no. 6, pp. 1365–1378, Nov. 2008. 5, 29

[17] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys, "Vision-Based Autonomous Mapping and Exploration using a Quadrotor MAV," in *IEEE Int. Conf. Intell. Robot. Syst.*, Oct. 2012, pp. 4557–4564. 5, 17, 37

[18] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A General Framework for Graph Optimization," in *IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 3607–3613. 5, 26, 29

[19] "ROS." [Online]. Available: http://www.ros.org 11

[20] R. Beard and T. McLain, *Small Unmanned Aircraft*. Princeton University Press, 2012. 13, 20, 24

[21] R. C. Leishman, J. C. MacDonald, R. W. Beard, and T. W. McLain, "Quadrotors and Accelerometers: State Estimation with an Improved Dynamic Model," *IEEE Control Syst.*, vol. 34, no. 1, pp. 28–41, 2014. 14, 20

[22] R. C. Leishman and T. W. McLain, "A Multiplicative Extended Kalman Filter for Relative Rotorcraft Navigation," *IEEE Trans. Robot.*, pp. 1–11, 2013. 18

[23] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, "Multi-Sensor Fusion for Robust Autonomous Flight in Indoor and Outdoor Environments with a Rotorcraft MAV," in *IEEE Int. Conf. Robot. Autom.*, Hong Kong, China, 2014, pp. 4974–4981. 18, 21, 37

[24] J. Zhang, M. Kaess, and S. Singh, "Real-Time Depth Enhanced Monocular Odometry," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2014, pp. 4973–4980. 20

[25] D. Scaramuzza and F. Fraundorfer, "Visual Odometry: Part I: The First 30 Years and Fundamental," *IEEE Robot. Autom. Mag.*, vol. 18, no. 4, pp. 80–92, 2011. 22

[26] F. Fraundorfer and D. Scaramuzza, "Visual Odometry: Part II: Matching, Robustness, Optimization, and Applications," *IEEE Robot. Autom. Mag.*, vol. 19, no. 2, pp. 78–90, 2012. 22

[27] Z. Fang and S. Scherer, "Experimental Study of Odometry Estimation Methods using RGB-D," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2014, pp. 680–687. 22

[28] J. Ferrin, R. Leishman, R. Beard, and T. McLain, "Differential Flatness Based Control of a Rotorcraft for Aggressive Maneuvers," in *IEEE Int. Conf. Intell. Robot. Syst.*, Sep. 2011, pp. 2688–2693. 23

[29] J. Sivic and A. Zisserman, "Efficient Visual Search for Objects in Videos," *IEEE Conf. Robot. Autom.*, vol. 96, no. 4, pp. 548–566, 2008. 26

[30] M. Cummins and P. Newman, "FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance," pp. 647–665, 2008. 26

[31] "QUT Robotics for the Real World." [Online]. Available: https://wiki.qut.edu.au/display/cyphy/Robotics@QUT 26

[32] F. Dellaert and M. Kaess, "Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing," pp. 1181–1203, Dec. 2006. 29

[33] A. Chambers, S. Scherer, L. Yoder, S. Jain, S. Nuske, and S. Singh, "Robust Multi-Sensor Fusion for Micro Aerial Vehicle Navigation in GPS-Degraded/Denied Environments," in *Am. Control Conf.*, 2014, pp. 1892–1899. 38

[34] J. G. Rogers, J. R. Fink, and E. A. Stump, "Mapping with a Ground Robot in GPS Denied and Degraded Environments," in *Am. Control Conf.*, 2014, pp. 1880–1885. 38

[35] J. Rehder, K. Gupta, S. Nuske, and S. Singh, "Global Pose Estimation with Limited GPS and Long Range Visual Odometry," in *IEEE Int. Conf. Robot. Autom.*, May 2012, pp. 627–633. 38, 41

[36] R. Morton and E. Olson, "Robust Sensor Characterization via Max-Mixture Models: GPS Sensors," in *IEEE Int. Conf. Intell. Robot. Syst.*, Nov. 2013, pp. 528–533. 39

[37] Y. Latif, C. Cadena, and J. Neira, "Robust Loop Closing Over Time for Pose Graph SLAM," *Int. J. Rob. Res.*, vol. 32, no. 14, pp. 1611–1626, 2013. 39

[38] N. Sunderhauf and P. Protzel, "Switchable Constraints for Robust Pose Graph SLAM," in *IEEE Int. Conf. Intell. Robot. Syst.*, Oct. 2012, pp. 1879–1884. 39

[39] N. Sunderhauf and P. Protzel, "Towards a Robust Back-End for Pose Graph SLAM," in *IEEE Int. Conf. Robot. Autom.*, May 2012, pp. 1254–1261. 39

[40] N. Sunderhauf and P. Protzel, "Switchable Constraints vs. Max-Mixture Models vs. RRR - A Comparison of Three Approaches to Robust Pose Graph SLAM," in *IEEE Int. Conf. Robot. Autom.*, May 2013, pp. 5198–5203. 40

[41] P. Agarwal, G. D. Tipaldi, L. Spinello, C. Stachniss, and W. Burgard, "Robust Map Optimization using Dynamic Covariance Scaling," in *IEEE Int. Conf. Robot. Autom.*, May 2013, pp. 62–69. 40, 46

[42] N. Sunderhauf, M. Obst, G. Wanielik, and P. Protzel, "Multipath Mitigation in GNSS-Based Localization Using Robust Optimization," in *IEEE Intell. Veh. Symp.*, Jun. 2012, pp. 784–789. 40, 41

## APPENDIX A. FORMULATING GRAPH-BASED LEAST-SQUARES OPTIMIZATION

The goal of pose-graph optimization is to find the arrangement of poses that most likely results from a given set of edges. In other words, the set of nodes $\mathbf{x}$ that maximizes $p(\mathbf{x} \mid \mathbf{z})$ where $\mathbf{z}$ is a set of edges connecting the nodes. However, because there is noise in every measurement $\mathbf{z}_k$, this is an impossible task. There is another related distribution that we are able to compute, that is the probability of a measurement given a vehicle state, $p(\mathbf{z}_k \mid \mathbf{x})$. This is known as our sensor or observation model. We assume $\mathbf{x}$ is known. Noting that the measurements are independent, given the state we can arrive at the following

$$p(\mathbf{z}_{1:K} \mid \mathbf{x}_{1:N}) = \prod_{k=1}^{K} p(\mathbf{z}_k \mid \mathbf{x}_{1:N})$$

Using Bayes' rule, we can rewrite the original distribution $p(\mathbf{x} \mid \mathbf{z})$

$$p(\mathbf{x} \mid \mathbf{z}) = \frac{p(\mathbf{z}_{1:K} \mid \mathbf{x}_{1:N}) p(\mathbf{x})}{p(\mathbf{z})}$$

If we know nothing about the prior, $p(\mathbf{x})$ is a uniform distribution with constant value. We know the measurements $\mathbf{z}$, so the normalizer is a constant number. With that, we can conclude that

$$p(\mathbf{x}_{1:N} \mid \mathbf{z}_{1:K}) \propto \prod_{k=1}^{K} p(\mathbf{z}_k \mid \mathbf{x}_{1:N}).$$

Assuming that the measurements are affected by zero-mean normal noise, the likelihood will also be Gaussian with mean $\hat{\mathbf{z}}$ and information matrix $\mathbf{\Omega}$. In SLAM, $\hat{\mathbf{z}}$ is the relative transformation between the two constrained poses. $\mathbf{\Omega}$ is the inverse of the measurement covariance. The resulting Gaussian takes the form

$$p(\mathbf{z}_k \mid \mathbf{x}) \propto \exp\left(-(\hat{\mathbf{z}}_k - \mathbf{z}_k)^T \mathbf{\Omega}_k (\hat{\mathbf{z}}_k - \mathbf{z}_k)\right).$$

We define $\mathbf{h}_k(\mathbf{x}) = \hat{\mathbf{z}}$, the sensor model, which is in general non-linear. We can linearize $\mathbf{h}_k(\mathbf{x})$ by using the Taylor approximation where $\breve{\mathbf{x}}$ is a good initial guess of $\mathbf{x}^*$

$$\mathbf{h}_x(\breve{\mathbf{x}} + \Delta\mathbf{x}) \simeq \mathbf{h}_k(\breve{\mathbf{x}}) + \left.\frac{\partial \mathbf{h}_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}} \cdot \Delta\mathbf{x}.$$

Defining an error function such that $\mathbf{e}_k(\mathbf{x}) = \mathbf{h}_k(\mathbf{x}) - \mathbf{z}_k$, and substituting the linearized formula for $\mathbf{h}$ into it results in a linear error function

$$e_k(\breve{\mathbf{x}} + \Delta\mathbf{x}) = \Delta\mathbf{x}^T \mathbf{H}_k \Delta\mathbf{x} - 2\mathbf{b}_k \Delta\mathbf{x} + \mathbf{e}_k^T \mathbf{\Omega}_k \mathbf{e}_k.$$

With these definitions, we can now find the arrangement of nodes that maximizes the posterior by finding

$$\begin{aligned} \mathbf{x}^* &= \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{z} \mid \mathbf{x}) \\ &= \underset{\mathbf{x}}{\operatorname{argmin}} \sum_{k=1}^{K} (\mathbf{h}_k(\mathbf{x}) - \mathbf{z}_k)^T \mathbf{\Omega}_k (\mathbf{h}_k(\mathbf{x}) - \mathbf{z}_k) \\ &= \underset{\mathbf{x}}{\operatorname{argmin}} \sum_{k=1}^{K} \mathbf{e}_k(\mathbf{x})^T \mathbf{\Omega}_k \mathbf{e}_k(\mathbf{x}). \end{aligned} \tag{A.1}$$

Using the previous definitions we can expand one summand of the objective function as

$$F_k(\breve{\mathbf{x}} + \Delta\mathbf{x}) = \Delta\mathbf{x}^T \mathbf{H}_k \Delta\mathbf{x} - 2\mathbf{b}_k \Delta\mathbf{x} + \mathbf{e}_k^T \mathbf{\Omega}_k \mathbf{e}_k.$$

Finally, summing over all measurements, we arrive at the quadratic objective function

$$F(\breve{\mathbf{x}} + \Delta\mathbf{x}) = \Delta\mathbf{x}^T \mathbf{H} \Delta\mathbf{x} - 2\mathbf{b} \Delta\mathbf{x} + \mathbf{c}$$

which can be minimized with the increments $\Delta\mathbf{x}^*$,

$$\mathbf{x}^* = \breve{\mathbf{x}} + \Delta\mathbf{x}^*.$$

Solving the objective yields the optimal increment $\Delta\mathbf{x}^*$ such that applied to $\mathbf{x}$ will be closer to $x^*$. The quadratic objective can be solved by taking the derivative with respect to $\Delta\mathbf{x}$, equating it to

zero, and solving the resulting equation.

$$\mathbf{H}\Delta\mathbf{x}^* = \mathbf{b}$$

The process of computing $\Delta\mathbf{x}^*$ and stepping toward the optimum is one iteration of the Gauss-Newton algorithm. Other related algorithms for stepping toward the optimal solution exist, but in this thesis Gauss-Newton is used for all pose-graph optimization unless otherwise stated.

### A.0.1 On the Structure of $\mathbf{J}, \mathbf{H}$ and $\mathbf{b}$

As described in [9], knowledge of the sparse structure of $\mathbf{J}, \mathbf{H}$ and $\mathbf{b}$ can aid in efficient methods to form the respective matrices. In the case of a general odometry constraint, the Jacobian is

$$\mathbf{J}_{ij} = \begin{pmatrix} \mathbf{0}\cdots\mathbf{0} & \mathbf{A}_{ij} & \mathbf{0}\cdots\mathbf{0} & \mathbf{B}_{ij} & \mathbf{0}\cdots\mathbf{0} \end{pmatrix}$$

where $\mathbf{A} = \frac{\partial \mathbf{e}_{ij}}{\partial \mathbf{x}_i}$ and $\mathbf{B} = \frac{\partial \mathbf{e}_{ij}}{\partial \mathbf{x}_j}$. The resulting contributions to the Hessian, $\mathbf{H}$, and vector $\mathbf{b}$ are

$$\mathbf{H}_{ij} = \begin{pmatrix} \ddots & & & & \\ & \mathbf{A}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{A}_{ij} & \cdots & \mathbf{A}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{B}_{ij} & \\ & \vdots & \ddots & \vdots & \\ & \mathbf{B}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{A}_{ij} & \cdots & \mathbf{B}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{B}_{ij} & \\ & & & & \ddots \end{pmatrix}$$

and

$$\mathbf{b}_{ij} = \begin{pmatrix} \vdots \\ \mathbf{A}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{e}_{ij} \\ \vdots \\ \mathbf{B}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{e}_{ij} \\ \vdots \end{pmatrix}.$$

We extend the approach of analysing the structure of $\mathbf{H}$ and $\mathbf{b}$ for the case of switching constraints. If we add switching constraints, the Jacobian becomes

$$\mathbf{J}_{ij} = \begin{pmatrix} \mathbf{0}\cdots\mathbf{0} & \mathbf{A}_{ij} & \mathbf{0}\cdots\mathbf{0} & \mathbf{B}_{ij} & \mathbf{0}\cdots\mathbf{0} & \mathbf{C}_{ij} & \mathbf{0}\cdots\mathbf{0} \end{pmatrix}$$

where $\mathbf{C} = \frac{\partial \mathbf{e}_{ij}}{\partial \mathbf{s}_{ij}}$. The resulting block matrix $\mathbf{H}$ and vector $\mathbf{b}$ are

$$\mathbf{H}_{ij} = \begin{pmatrix} \ddots & & & & & & \\ & \mathbf{A}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{A}_{ij} & \cdots & \mathbf{A}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{B}_{ij} & \cdots & \mathbf{A}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{C}_{ij} & \\ & \vdots & \ddots & \vdots & & \vdots & \\ & \mathbf{B}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{A}_{ij} & \cdots & \mathbf{B}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{B}_{ij} & \cdots & \mathbf{B}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{C}_{ij} & \\ & \vdots & & \vdots & \ddots & \vdots & \\ & \mathbf{C}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{A}_{ij} & \cdots & \mathbf{C}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{B}_{ij} & \cdots & \mathbf{C}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{C}_{ij} & \\ & & & & & & \ddots \end{pmatrix}$$

and

$$\mathbf{b}_{ij} = \begin{pmatrix} \vdots \\ \mathbf{A}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{e}_{ij} \\ \vdots \\ \mathbf{B}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{e}_{ij} \\ \vdots \\ \mathbf{C}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{e}_{ij} \\ \vdots \end{pmatrix}.$$

The addition of switching variables requires the addition of switching priors as well. Their contribution to the Hessian and $\mathbf{b}$ are

$$\mathbf{H}_{ij}^{sp} = \begin{pmatrix} \ddots & & & \\ & \mathbf{0} & \cdots & \mathbf{0} \\ & \vdots & \ddots & \vdots \\ & \mathbf{0} & \cdots & \frac{\partial \mathbf{e}_{ij}^{\text{sp}}}{\partial \mathbf{s}_{ij}}^T \mathbf{\Omega}_{ij}^{\text{sp}} \frac{\partial \mathbf{e}_{ij}^{\text{sp}}}{\partial \mathbf{s}_{ij}} \\ & & & \ddots \end{pmatrix}$$

and

$$\mathbf{b}_{ij} = \begin{pmatrix} \vdots \\ \mathbf{C}_{ij}^{\mathrm{sp}T} \mathbf{\Omega}_{ij}^{\mathrm{sp}} \mathbf{e}_{ij}^{\mathrm{sp}} \\ \vdots \end{pmatrix}.$$

The MATLAB simulation presented in this thesis (Section 2.5) uses the aforementioned structures to formulated the Hessian and **b** vector for traditional least-squares and switching constraint implementations.