Master's Thesis

Lidar-Visual-Inertial Odometry via Modifications and Improvements to Super Odometry

Andrew VanOsten

CMU-RI-TR-22-35

July 2022

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Prof. Michael Kaess, Chair
Prof. Sebastian Scherer
Sudharshan Suresh

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Copyright © 2022 Andrew VanOsten

Abstract

The main focus of this thesis involves improvements and extensions to Super Odometry [1], a preexisting method for lidar-inertial odometry. This was done in the context of the DARPA RACER program as a member of Carnegie Mellon's DEAD Fast team, aiming to provide reliable state estimation for an autonomous off-road ground vehicle. Super Odometry was modified to take simultaneous input from multiple Velodyne VLP-32C lidars, and a more complex "dewarping" method was added to fully account for motion of the vehicle during a given lidar scan. Significant effort was put into streamlining and optimizing the implementation of Super Odometry to maintain real-time performance and provide a stable baseline for future work on the RACER program. In addition, a secondary pose graph was added to fuse the odometry solution with absolute position data from GPS. This provided a full 6-DoF pose estimate in UTM coordinates to the rest of the vehicle's autonomy stack.

The second thrust of this work focused on merging TartanVO [2], a learning-based visual odometry method, with Super Odometry. By design, Super Odometry can easily accept another source of pose estimates in addition to lidar odometry. These poses are added as relative pose factors in its factor graph. Significant implementation efforts were put into producing a working lidar-visual-inertial odometry method under the Super Odometry framework.

Acknowledgments

Thank you to my advisor, Michael Kaess, for his guidance, for his patience, and for helping me find work at CMU that I genuinely enjoy. Despite my tendency to lean towards systems work rather than research, Michael has always supported me and looked out for my best interest. Thank you to the other members of the Robot Perception Lab, for tolerating my not-so-research-focused biweekly updates. Thank you to my teammates on the RACER project who gave me a community to spend far too much time with, whether we were spending a day at Gascola, or embarking on a two-week field test across the country. Thank you to my committee members, Sebastian Scherer and Sudharshan Suresh, for their willingness to review my work and provide their insights. I would also like to thank the SMART program under the U.S. Department of Defense for funding this degree.

I owe a big thank you to the people of St. Stephen's Episcopal Church in McKeesport: to Moni, the Pratts, Ethel, the Ochaps, the Lebeddas, and plenty of others. You all welcomed me from the moment I walked in the door, and you have given me a church family that feels like a home away from home. I will always be grateful for the time I have been able to spend with you during my time in Pittsburgh, and for the impact that you have had on my life.

Most importantly, thank you to my family: Mom, Dad, Carrie, and Zack. You all have always been there to support me, to watch out for me, to guide me, and to pick up the phone to talk for a few minutes, or a few hours. I love you all, and I cannot express how much you mean to me, or how much I appreciate everything you continue to do for me.

Contents

1 Introduction 1

2 Background 3
- 2.1 State Estimation for DARPA RACER . . . 3
- 2.2 Super Odometry . . . 3
  - 2.2.1 Feature Extraction and Lidar Odometry . . . 4
  - 2.2.2 IMU Odometry . . . 6
- 2.3 TartanVO . . . 8

3 Lidar-Inertial State Estimation 9
- 3.1 Improvements to Super Odometry . . . 9
  - 3.1.1 Point Cloud Dewarping . . . 9
  - 3.1.2 Fusing Point Clouds from Multiple Lidars . . . 11
  - 3.1.3 Initialization Procedure . . . 12
  - 3.1.4 Efficient Feature Map Representation . . . 13
  - 3.1.5 Attaining Real-Time Performance . . . 13
  - 3.1.6 Handling Real-World Timing Issues . . . 15
- 3.2 Global State Estimation: GPS Fusion . . . 15
  - 3.2.1 Motivation for a Global Pose Estimate . . . 15
  - 3.2.2 Pose-Graph Structure . . . 16

4 Lidar-Visual-Inertial State Estimation 17
- 4.1 High-Level Information Flow . . . 17
- 4.2 Factor Graph Structure . . . 17
- 4.3 Implementation Notes . . . 18

5 Results 21

6 Future Work 25
- 6.1 Learning-Based Methods for Uncertainty Estimation . . . 25
- 6.2 Correcting for Orientation Drift . . . 26
- 6.3 Maintaining Performance at Higher Speeds . . . 27

7 Conclusion 29

List of Figures

2.1 Image of a vehicle used on the DARPA RACER program. 4
2.2 Overview of Super Odometry information flow. 4
2.3 Low-level flowchart for Super Odometry. 5
2.4 Super Odometry factor graph, original full-smoothing approach. 7

3.1 Timing diagram for point cloud dewarping. 10
3.2 Example motion that can occur during a lidar scan. 10
3.3 Timing diagram for fusing point clouds from two lidars. 12
3.4 Super Odometry factor graph, fixed-lag smoothing approach. 14
3.5 GPS fusion pose graph. 16

4.1 Overview of Super Odometry information flow with visual odometry. 17
4.2 Super Odometry factor graph with binary pose factors from TartanVO. . . . 18
4.3 Adding a unary factor to the Super Odometry factor graph. 19

5.1 Super Odometry result showing drift along 3.4 km of trails (Course A). . . . 22
5.2 Super Odometry result showing drift over 2.75 km in an open area (Course B). 22
5.3 Registered point cloud from lidar-inertial Super Odometry with a single lidar. 23
5.4 Registered point cloud from lidar-inertial Super Odometry with three lidars. 23
5.5 Isometric view, odometry in an open area. 24
5.6 Top-down view, odometry on a narrow trail. 24

6.1 Framework considered for learning uncertainty values for pose estimates. . . 26

Chapter 1

Introduction

At its core, this work focuses on the problem of state estimation using lidar, vision, and inertial sensors. The systems described in this document were developed for Carnegie Mellon's DEAD Fast team working on the DARPA RACER program. Although development has focused on autonomous ground vehicles, the concepts described below are not limited to these platforms. State estimation plays a critical role in an autonomous system; planners must know the vehicle's current pose in order to plan towards a goal, and controllers need this pose to be able to correct for tracking errors. In addition, lidar data is much more useful when it can be transformed into a non-moving coordinate frame. This task relies heavily on the state estimate and allows for point clouds to be accumulated over time, producing a denser and more detailed map. Issues or failures in state estimation can quickly lead to failure of the entire system, so robustness and reliability are critical.

The starting point for this work is Super Odometry [1], a multi-sensor method which fuses integrated IMU measurements with input from other odometry sources. Chapter 2 describes the baseline lidar-inertial version of Super Odometry, and chapter 3 covers the changes and improvements that were made to it under the RACER program. Chapter 4 discusses the addition of TartanVO [2], a learning-based visual odometry method, and some of the challenges associated with this effort.

Chapter 2

Background

2.1 State Estimation for DARPA RACER

DARPA's Robotic Autonomy in Complex Environments with Resiliency (RACER) is a U.S. Department of Defense program working towards high-speed, off-road autonomous driving. This program aims to produce unmanned ground vehicles capable of operating "at the limit of the vehicle's mechanical systems and at, or beyond, human-driven speeds and efficiencies" [3]. One of the vehicles used on this project is shown in figure 2.1. As a portion of a larger autonomy software stack, the primary goal of the state estimation system is to estimate the vehicle's position, orientation, and velocity. In this context, this is a challenging task—high speeds, aggressive maneuvers, and significant vibrations all complicate the process. The vehicle includes several lidars, stereo camera pairs, and inertial measurement units (IMUs), all of which can contribute to state estimation.

2.2 Super Odometry

This section is intended to provide an overview of Super Odometry, which is central to the rest of this document. Of course, more information can be found in the original paper [1]. The idea at the core of this method is to center the state estimation process around the integration of IMU measurements. Even a relatively inexpensive IMU can provide an adequate, high-frequency estimate of pose and velocity over short periods of time. However, an IMU-only method is likely to diverge after only a few seconds as any noise from the sensor is integrated. In contrast, lidar odometry methods are often capable of producing a reliable pose estimate based on nearby geometric features, but at a much lower frequency and with more latency.

By design, Super Odometry chains these two types of systems together, using slow but reliable pose estimates from lidar odometry to constrain an IMU-based method. This combines the robustness of lidar-based methods with the high update rate and low latency that are possible using an IMU. The high-level flow of information in Super Odometry can be seen in figure 2.2. Lidar and IMU odometry nodes operate separately, and only pose predictions are communicated between the two. The lidar odometry node uses the IMU-based odometry only as an initialization point for its point cloud registration. After aligning a point cloud to

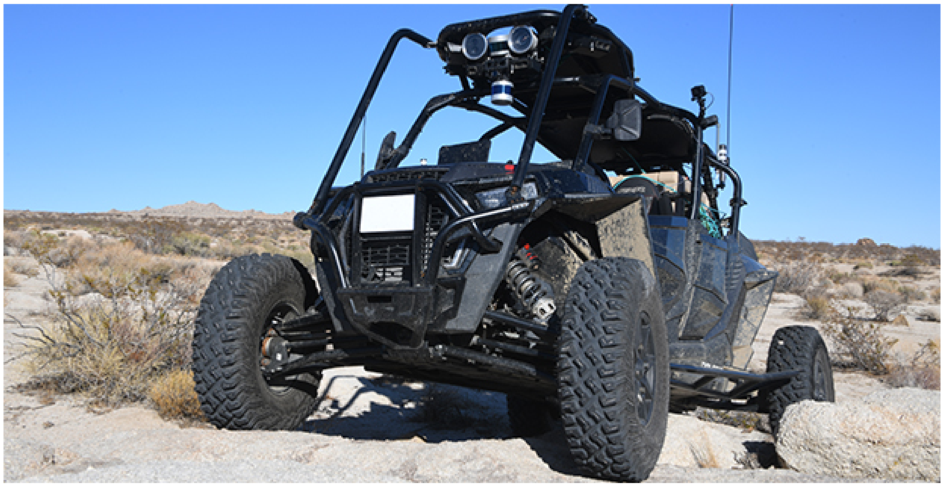

Figure 2.1: Image of a vehicle used on the DARPA RACER program. Image from [3].

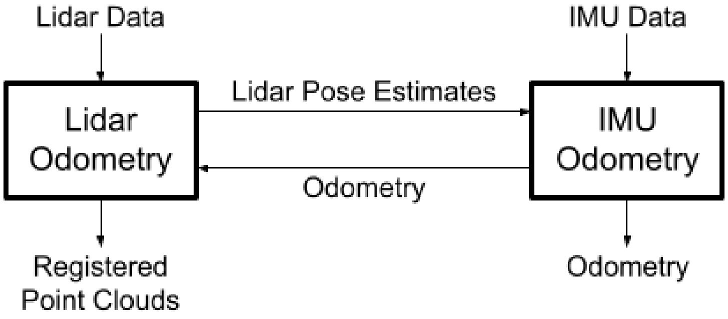

Figure 2.2: Overview of Super Odometry information flow.

an internally-maintained map, an entirely lidar-based pose estimate is produced. This pose is then sent to the IMU odometry node and incorporated into the factor graph. As covered in [1], this structure makes adding additional odometry sources fairly straightforward. For example, pose estimates from visual-inertial odometry would simply act as additional pose factors in the graph, just like those produced by lidar odometry.

A more detailed depiction of Super Odometry is shown in figure 2.3. Each of the submodules shown in this diagram will be discussed in more depth below.

2.2.1 Feature Extraction and Lidar Odometry

Parsing Lidar Scans

When data is initially received from the lidars (the Velodyne VLP-32C was used here), it is stored in a very information-dense format. This is efficient for transport, but is not immediately useful for further processing. This parsing step extracts the location of each lidar return in spherical coordinates, and transforms it into Cartesian coordinates relative to the center of the lidar. Points are then transformed into a shared reference frame and repackaged into a more user-friendly data structure. From this point on, the `pcl::PointCloud` class

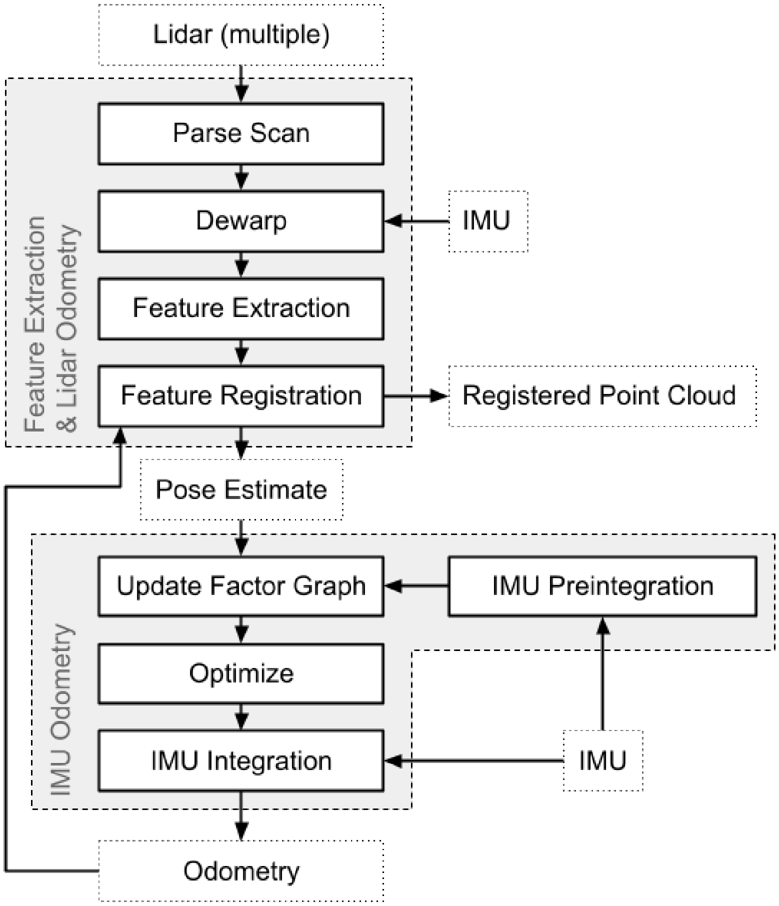
Lidar (multiple)
Feature Extraction & Lidar Odometry
Parse Scan
Dewarp
IMU
Feature Extraction
Feature Registration
Registered Point Cloud
Pose Estimate
IMU Odometry
Update Factor Graph
IMU Preintegration
Optimize
IMU Integration
IMU
Odometry

Figure 2.3: Low-level flowchart for Super Odometry.

from the Point Cloud Library [4] is used to store and manipulate point clouds. Unwanted points are also removed at this stage, including any returns that bounced off of the vehicle itself, and extremely low-intensity returns from the vehicle's surroundings.

Point Cloud Dewarping

After the initial parsing step, point clouds are passed through a "dewarping" process. Although all points are already described in the same reference frame, this frame moves with the vehicle and is therefore different at any given timestep relative to a non-moving observer. The dewarping process transforms all the points in a given cloud so that they are described in the same reference frame, *at the same moment in time*. In essence, this step accounts for the motion of the vehicle that occurred while the points in this cloud were measured; this motion is estimated by integrating IMU measurements. This process is discussed in much greater detail in section 3.1.1.

Feature Extraction

Next, geometric features—edges and planes—are extracted from the point cloud. This step attempts to find features which stand out, and may be identifiable in past or future point clouds. The number of selected features is generally much lower than the total number of points in the cloud, which reduces the computational requirements for the registration step, and allows the system to run in real-time.

Feature Registration

At this point, the newly-extracted set of features can be aligned to a map of previously-seen features to produce a pose estimate. This local map is simply an accumulation of features from past scans. Registration is set up as a nonlinear least-squares problem, and solved iteratively using the Ceres Solver [5]. The pose estimate is sent on to the IMU odometry node, and a registered point cloud is produced. Note that this registered cloud includes the full set of points from the output of the dewarping step, and not just the feature points. After registration, the features are added to the map, and downsampling is performed to keep the map from growing too large.

2.2.2 IMU Odometry

Factor Graph Update and Optimization

When a pose estimate is received from the feature registration submodule, it is added as a unary pose factor in the factor graph shown in figure 2.4. The red "preintegrated IMU factors" in the figure are produced by integrating IMU data as it is received from the sensor, as described in [6]. The blue factors are zero-mean priors on the initial velocity and bias variables, and the orange "constant bias factors" assert that, with some uncertainty, the IMU biases should not change from one timestep to the next. Collectively, these factors constrain the new variables that are added to the graph at this step. These variables make up the state vector, which includes 15 degrees of freedom: the pose and linear velocity of the vehicle, as

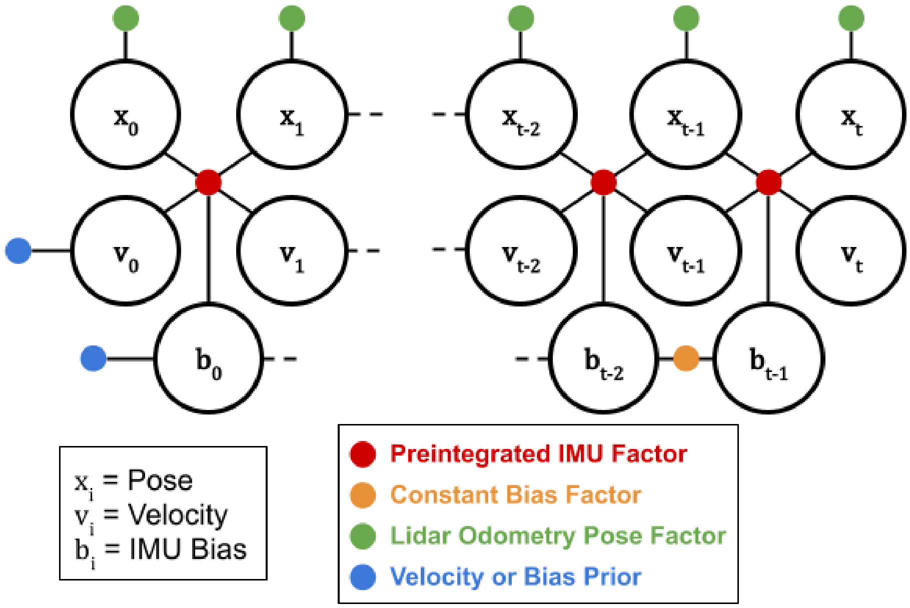

Figure 2.4: Super Odometry factor graph, original full-smoothing approach.

well as bias values for the accelerometer and gyroscope. After the graph has been updated, values for all variables in the graph are estimated using the iSAM2 incremental solver [7]. The code to maintain the factor graph and solve for a state estimate is implemented using the GTSAM library. The original implementation used a full-smoothing approach, where old factors and variables are never removed from the graph. This can slow down the optimization process as the graph grows and, when run for a long duration, threaten the ability of the system to maintain real-time performance. A solution to this issue is discussed in section 3.1.5.

IMU Integration

It should be noted that figure 2.3 contains separate submodules labeled “IMU Integration”, and “IMU Preintegration”. While these are performing what is essentially the same computation, they serve two distinct purposes. The IMU preintegration step integrates the IMU measurements that occur between lidar-based pose estimates in the factor graph. This provides both the inertial factor which is added to the graph, and an estimate of pose and velocity which is used as an initialization point for the newly added variables. In contrast, the IMU integration step integrates from the most recent variables in the graph, up to the latest IMU measurement, which is potentially much more recent. Its purpose is to produce a state estimate at a high frequency and with very low latency. Updates to the factor graph occur with a relatively high latency and low frequency. However, by constantly integrating IMU measurements as they are received, pose and velocity estimates can be produced at the full update rate of the IMU and with extremely low latency.

2.3 TartanVO

TartanVO [2] is a learning-based method for visual odometry. Given the camera's intrinsics and a consecutive pair of images, a convolutional neural network is used to regress directly to an estimate of the translation and rotation of the camera which occurred between the two frames. TartanVO was trained on synthetic data from the TartanAir dataset [8], but has been shown to generalize well to real-world systems. Chapter 4 of this document describes a method for fusing this visual odometry solution with Super Odometry.

Chapter 3

Lidar-Inertial State Estimation

3.1 Improvements to Super Odometry

With the goal of producing a robust and stable state estimation system for the DARPA RACER project, a number of changes and improvements were made to the original version of Super Odometry which was described in the previous chapter. This section describes the most prominent changes that were made to the system.

3.1.1 Point Cloud Dewarping

A note on notation: In this section as well as section 3.1.2, transforms will be referred to as a single variable T, with subscripts and superscripts denoting the frames in question. These transforms can be thought of as homogeneous transformation matrices in $SE(3)$, but they have been implemented as the combination of a translation vector $\mathbf{t} \in \mathbb{R}^3$, and a rotation represented by a unit quaternion q.

When a point cloud is received from the lidar, points are represented in spherical coordinates as an azimuth, elevation, and range. Consider a point cloud $\mathcal{P}$, where each of these points has been converted to Cartesian coordinates relative to the "odometry" coordinate frame. This is the common reference frame used for all point clouds and odometry to avoid unnecessary transforms. Since the lidar is constantly spinning and taking measurements, a point cloud is really just a collection of lidar returns from some arbitrary time window. Each point in the cloud was measured at a slightly different moment in time, and each point is described relative to the odometry frame *at the time when that point was measured.* Because the lidar itself is moving with the vehicle, every point in the cloud is described relative to a slightly different reference frame. This timing is shown in figure 3.1, with each point in the cloud having a different timestamp and therefore a different frame of reference. "Dewarping" is the process of transforming these points into a common reference frame so that they can be registered to the map collectively. Specifically, this common reference frame is defined as the odometry frame at time t_{start}, where t_{start} is the timestamp for the beginning of the current lidar scan (that is, the timestamp for the earliest point in the cloud). An example of the geometry of the dewarping process is shown in figure 3.2. Before dewarping, the point p_i is described relative to the odometry frame at the corresponding time t_i. The goal of

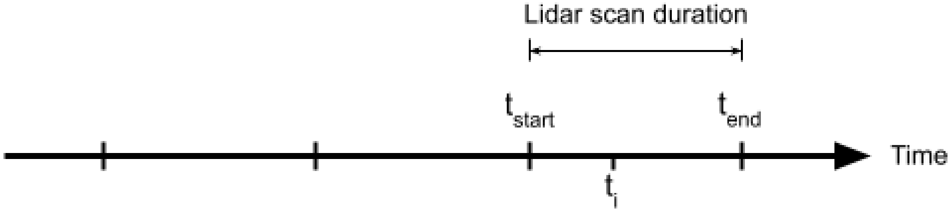

Figure 3.1: Marks on the time axis indicate the divisions between lidar “scans”. The time t_i indicates the timestamp for the i-th point in the cloud. The aim of the dewarping step is to transform all points such that they are described relative to the location of the odometry coordinate frame at time t_{start}.

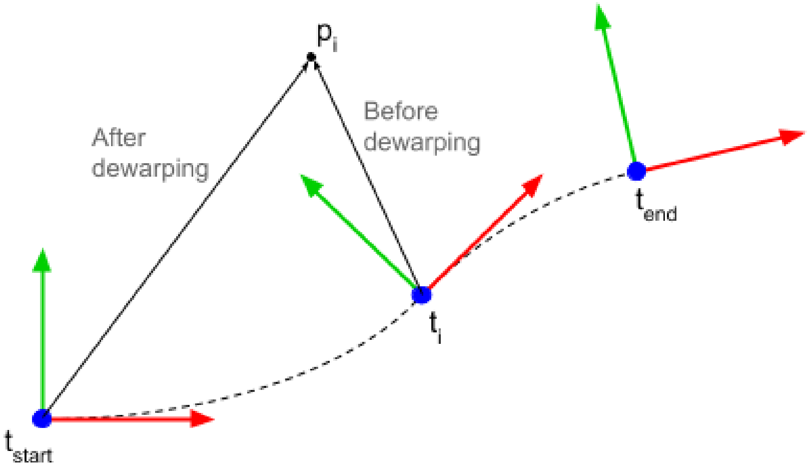

Figure 3.2: This diagram shows an example motion that could occur during a lidar scan. Before dewarping, the i-th point p_i is described relative to the odometry frame at the corresponding timestamp t_i. After dewarping, all points in the cloud are described relative to the odometry frame at the start time t_{start}.

dewarping is to describe p_i, and all other points in the cloud, with respect to the odometry frame at time t_{start}.

While the original version of Super Odometry did include a dewarping step (as mentioned in section 2.2.1), that implementation only accounted for the rotation of the vehicle and not its translation. In the context of the DARPA RACER program, the vehicle is often moving at high speed along a relatively straight path (over short time intervals), so compensating for the translation component is critical to producing an accurate point cloud. This implementation accounts for changes in the full pose of the vehicle using both the accelerometer and the gyroscope. IMU measurements are integrated using the GTSAM implementation of the methods from [6] and, when combined with velocity and IMU bias estimates from the IMU odometry node, an estimate of the full 6 DoF pose of the vehicle can be produced. By interpolating between these pose estimates, the pose of the IMU can be estimated for any time t_i relative to the start time t_{start}. This pose is referred to as $T^{\text{IMU}}_{(t_{\text{start}},t_i)}$ below. Because the involved time intervals are small, it is assumed that pure IMU integration will provide an adequate pose estimate.

If the given point cloud $\mathcal{P}$ corresponds to a time interval $[t_{\text{start}}, t_{\text{end}}]$, then this dewarping process requires an estimate of the motion of the odometry frame throughout this interval. If the i-th point in the point cloud corresponds to a timestamp t_i, then one must estimate the pose of the odometry frame at time t_i relative to the pose of the same frame at time t_{start}. We will denote this transform $T^{\text{odom}}_{(t_{\text{start}},t_i)}$. By integrating IMU data as described above, we can estimate the motion of the IMU over the same time period: $T^{\text{IMU}}_{(t_{\text{start}},t_i)}$. Combining this information with the known, static transform from the IMU to the odometry frame ($T^{\text{odom}}_{\text{IMU}}$), the desired transform can be computed as follows.

$$T^{\text{odom}}_{(t_{\text{start}},t_i)} = \left(T^{\text{odom}}_{\text{IMU}}\right)\left(T^{\text{IMU}}_{(t_{\text{start}},t_i)}\right)\left(T^{\text{odom}}_{\text{IMU}}\right)^{-1}$$

Although the notation here is a bit dense, the concept is straightforward: estimate the pose of the odometry frame at some given time, relative to where it was at the “start” time. Recalling that the i-th point in the point cloud is described relative to the odometry frame at time t_i, the above transforms can be used to transform all points into the odometry frame at time t_{start}. This dewarped point cloud now represents all points in a single reference frame, so they can later be registered to the map collectively to produce an estimate of the pose at time t_{start}.

3.1.2 Fusing Point Clouds from Multiple Lidars

In order to increase the number of geometric features that are available for registration, points from multiple lidars are fused into a single cloud. The challenge presented by this merger is very similar to the dewarping problem described above: all points must be represented in the same reference frame, at the same moment in time. In fact, these processes are so closely related that they are implemented together in a single function. The core difference in the multi-lidar case is the introduction of a reference time t_{ref}. A timing diagram for this scenario is shown in figure 3.3. For all lidars, points are transformed such that they are described relative to the odometry frame at time t_{ref}. Critically, all lidars will use the same reference time, which is defined as the start time for the primary lidar. For each lidar, this introduces an additional transform to account for the motion that occurs between t_{start} and t_{ref}. In the end, the dewarping process is very similar to single-lidar dewarping and the final transforms are computed as follows.

$$T^{\text{odom}}_{(t_{\text{ref}},t_i)} = \left(T^{\text{odom}}_{\text{IMU}}\right)\left(T^{\text{IMU}}_{(t_{\text{start}},t_{\text{ref}})}\right)^{-1}\left(T^{\text{IMU}}_{(t_{\text{start}},t_i)}\right)\left(T^{\text{odom}}_{\text{IMU}}\right)^{-1}$$

Transforms between IMU frames at different points in time are once again computed by integrating IMU data and interpolating the result.

Using this structure, point clouds from multiple lidars can be dewarped using equivalent processes. By using the same t_{ref} for dewarping all lidars, all points are transformed into a common reference frame, at a common reference time. At that point, the clouds can be merged into a single data structure and treated as though all points were measured by the same sensor. This process repeats when a new scan is received from the primary lidar: a

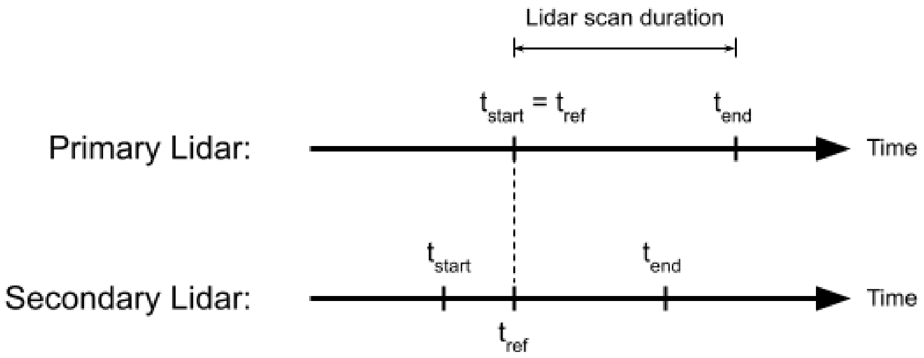

Figure 3.3: This diagram shows the timing of the dewarping process that is applied to two lidars to allow them to be fused. The scan from the primary lidar is dewarped normally, with the reference time set as the start time of the scan. The secondary lidar is dewarped such that its reference time is the same as the reference time used for dewarping the primary lidar. In this way, points from all lidars can be described in the same set of coordinates.

new t_{ref} is selected, and and the most recent scans from all lidars are merged into a single cloud.

3.1.3 Initialization Procedure

Dewarping on Initialization

Although the dewarping process described in section 3.1.1 has a very positive effect on the quality of registered point clouds and odometry, it is reliant on an accurate prediction of the motion of the lidar using IMU integration. This, in turn, is dependent on an accurate estimate of the IMU biases. This estimate is generally available from the IMU odometry node, but it tends to be briefly inaccurate on initialization. This would sometimes cause the system to fail entirely on initialization, because the dewarped point clouds were substantially inaccurate.

The solution to this is to briefly disable the dewarping step while the rest of the system is initialized. Other parts of the system assume that the vehicle is stationary on start-up, so the dewarping step is unnecessary at this stage regardless. Once the factor graph has incorporated a certain number of pose updates, it is assumed that the estimate of IMU biases has stabilized. Initialization is deemed to be complete, and the dewarping step is enabled.

Gravity-Aligned Initialization

One feature of the original version of Super Odometry was the ability to initialize its reference frame such that its x-y plane was normal to the direction of gravity. This allows for the orientation portion of odometry to be treated as an absolute estimate of orientation, rather than being relative to some arbitrary initial orientation. While this feature worked, its implementation needed some improvements. Previously, the lidar odometry and IMU odometry nodes each initialized separately, and it was assumed that they would choose essentially the same set of coordinates. To enforce this condition, the initialization procedure was reworked. Now, the IMU odometry node handles this initial orientation estimate, and the lidar odometry node must wait for this information to arrive, thus ensuring that the entire system agrees on the same set of coordinates. This approach also makes the addition of visual odometry more straightforward—the visual odometry node can simply wait for an initial orientation estimate, and use that to initialize its frame of reference.

3.1.4 Efficient Feature Map Representation

Part of Super Odometry's lidar odometry node is the "local map", which maintains a subset of previously-observed geometric features. When features are extracted from a new point cloud, they are aligned to this map to produce a pose estimate. The newly-aligned features are then added to the map to provide context for the next iteration of the process. Although the local map is constantly downsampled, it still contains a large amount of information, so its implementation can have a significant effect on performance. The representation used in the original version of Super Odometry was already quite efficient. Features were divided into large voxels (30 meter cubes) and each voxel contained both a point cloud and a KD-tree for fast spatial lookup. Nearest-neighbor searches are a large part of the registration process, so fast searches in the KD-trees are critical. The full map was stored as a fixed-size array of these voxels representing a cuboidal region. This approach allowed for fast indexing, but its major drawback came when the vehicle approached the edge of the map. As the name implies, the local map only maintains information for a limited region, and this region must be shifted when its edge is reached. With the array-of-voxels implementation, every voxel would be shifted by one unit to keep the vehicle from straying off the map. This meant that essentially every voxel in the map would have to be moved to a different position in the array, leading to a large amount of copying.

To avoid this slow and cumbersome process, the implementation of the local map was reworked. The idea of maintaining a collection of voxels was maintained, but the array of voxels was exchanged for a hash table. With this structure, a spatial coordinate can still be quickly mapped to a voxel, but the voxels do not need to be stored in a specific order in memory. More importantly, the region represented by the map is no longer a fixed cuboid. When the edge of the map is approached, new voxels are allocated and added to the map. To enforce a maximum size to the map, each voxel maintains a "last-modified" timestamp, which is updated whenever a feature is added to that voxel. When the maximum number of voxels is reached, the oldest voxels in the map are removed to maintain the maximum size requirement. This process does require iterating over all voxels to check their timestamps, but overall this should still be much more efficient than the previous method.

3.1.5 Attaining Real-Time Performance

During the development of all of the above modifications to Super Odometry, significant efforts went towards minimizing the latency of the produced odometry and registered point clouds. Because state estimation is often the first stage of an autonomy stack, any latency

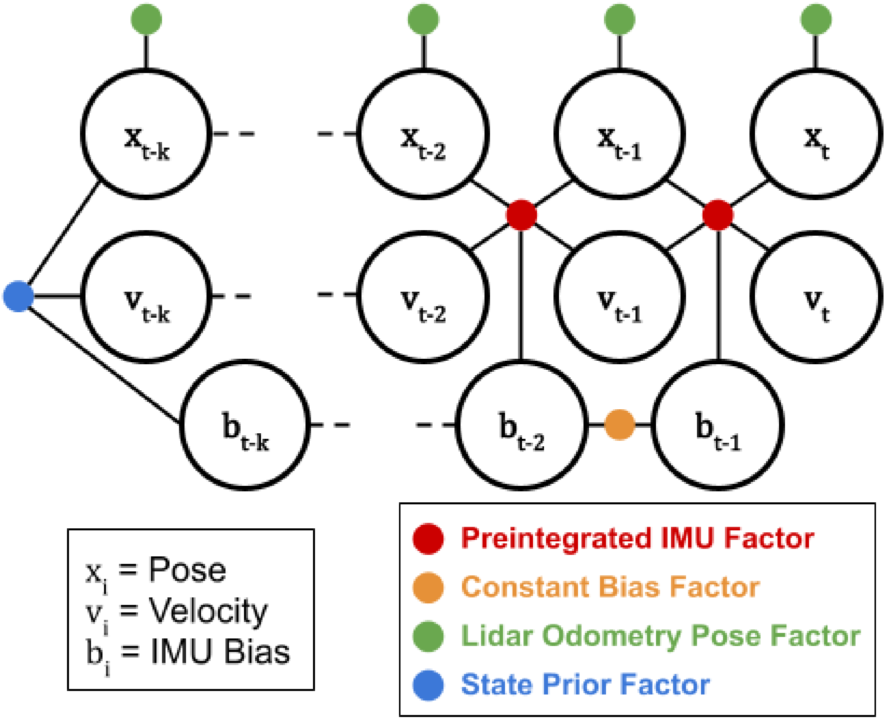

Figure 3.4: Super Odometry factor graph, fixed-lag smoothing approach.

cascades into modules that consume the state estimate. Large portions of the code base were refactored, optimized, and streamlined to make them as efficient as possible. This section details some specific efforts in this area.

Fixed-Lag Smoothing

As mentioned in section 2.2.2, the original version of Super Odometry used a full-smoothing approach for its factor graph, where old information is never removed from the graph. While this has the benefit of retaining all relevant information, the amount of time needed to optimize over the ever-growing graph can eventually become too large. One solution to this is a fixed-lag smoother, which puts a limit on the maximum size of the graph. When a new set of variables is added to the graph, the oldest set of variables is marginalized and replaced with a prior factor. The resulting factor graph, shown in figure 3.4, is a chain containing factors and variables from the last k timesteps. The cost of updating this graph remains relatively constant over time, meaning the system can run perpetually in real-time. For simplicity, the incremental iSAM2 solver was replaced by a standard Levenberg-Marquardt batch solver. Just as before, the fixed-lag smoother was implemented using GTSAM.

Multithreading Portions of the Code

With the goal of minimizing latency, multithreading can be a very useful tool in certain contexts. Many parts of the lidar odometry portion of Super Odometry involve applying some operation to thousands of points or features. In many cases these operations are independent of each other, making them a good candidate for multithreading. Most of these were implemented as some form of a parallelized `for` loop. Strict limits were applied on the maximum number of threads used, as too many threads often led to slowdowns as a result of the overhead required for coordination. In some cases, a single thread proved to be the best method. For example, applying a rigid-body transform to all points in a point cloud could be done on one thread in a fraction of a millisecond; multithreading this process would have yielded little (if any) gain.

Multiple threads were also used in the IMU odometry node, albeit in a very different way. In terms of major computations, this node had two goals. The first was to update the factor graph when a new lidar-based pose estimate was received. This could be a slow process, but it only needed to run at the relatively low update rate of the lidar. The node's second main purpose was to handle incoming IMU data, integrating each measurement as it was received. This was a fairly quick process, as it had to keep up with the high-frequency updates from the IMU. This step allowed for odometry updates to be produced at the full update rate of the IMU. To handle these dual goals, the computations were split across two separate threads. With careful handling of thread safety, the high-frequency odometry updates could continue in the background, even when the factor graph was in the process of being updated.

3.1.6 Handling Real-World Timing Issues

One of the major challenges of applying the above methods to a real autonomous system was dealing with the realities of sensors and timing issues. Checks were added to notice if a sensor stopped working, or if timestamps on sensor data jumped forward or backward in time. Because the expected update rates of the sensors were known, it was generally possible to handle a time jump (sudden change in timestamp) and continue operating. However, it was nearly impossible to run the system when sensors disagreed on the current time. This made synchronization between lidars and the IMU infeasible, and often caused the state estimation system to crash. A good area of future work would be to detect the erroneous sensor and predict what its timestamp *should be*, until it recovers and returns to functioning normally.

3.2 Global State Estimation: GPS Fusion

3.2.1 Motivation for a Global Pose Estimate

Up to this point, most of this document has discussed odometry. Generally speaking, odometry is a relative measure—it describes the current pose of the vehicle with respect to some arbitrary initial pose. This is sufficient for most parts of an autonomy stack, but there is one area where more information is needed. If the vehicle needs to deal with absolute location information, such as navigating to a predetermined set of coordinates, then it needs to relate its local odometry to an absolute frame of reference. More specifically, a means of fusing GPS with odometry is needed. For the DARPA RACER program, this was done by a standalone module downstream of Super Odometry using a pose graph.

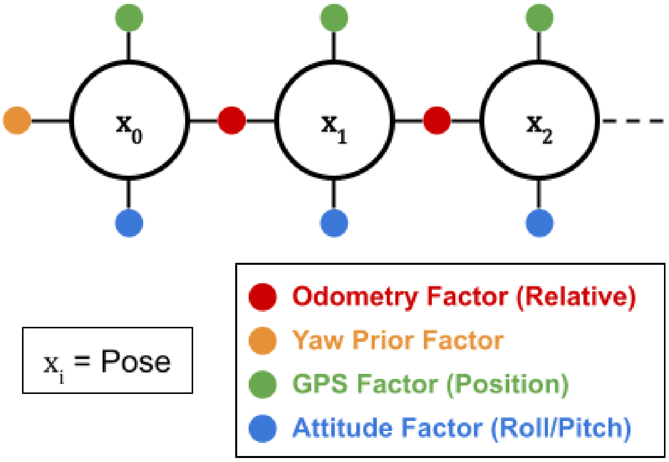

Figure 3.5: GPS fusion pose graph.

3.2.2 Pose-Graph Structure

The pose graph created for this task is shown in figure 3.5. The graph itself is relatively straightforward, though there is some nuance in appropriately constraining orientation, as this is not directly provided by GPS. GPS factors (green) are raw GPS information, and only provide information on the 3 DoF position of the pose variable x_i. Odometry factors come from the output of Super Odometry, and describe the 6 DoF change in pose during the period between GPS updates. The attitude and yaw factors come from an inertial navigation system (INS) which is running onboard the GPS module. The attitude factors apply only to roll and pitch, and are added to every pose variable. The yaw prior is added only to the first pose variable, and it has a high uncertainty associated with it. Before the vehicle has moved, this prior provides the only information on the vehicle's yaw in the global frame of reference. After some translation has occurred, the displacement between GPS positions will begin to constrain yaw, and the optimization can converge to the appropriate values.

Chapter 4

Lidar-Visual-Inertial State Estimation

4.1 High-Level Information Flow

As discussed in [1], Super Odometry is designed to allow for multiple odometry sources. Primarily, this would mean adding a visual odometry method to be fused with lidar odometry. The flow of information for this case is shown in figure 4.1. TartanVO [2] was used to provide 6 DoF pose estimates, which provided additional information to the factor graph in the IMU odometry node.

4.2 Factor Graph Structure

In initial implementations, pose estimates from visual odometry were added to the factor graph from figure 3.4 as unary pose priors, just like the factors from lidar odometry. This was an imperfect approach, and it failed quickly in practice. Just like any odometry, the visual and lidar odometry sources have some amount of drift in their pose estimates. When both types of pose information are added to the graph as "absolute" estimates (unary factors), any drift is accumulated and not accurately represented by the fixed uncertainty associated with these factors. The two odometry methods eventually disagree drastically, and the fused odometry result becomes somewhat worthless.

In reality, odometry is more like a *relative* estimate of pose, generating the change in pose between timesteps rather than an absolute measure. With this in mind, a better approach would be to replace the unary pose factors with binary factors connecting pairs of

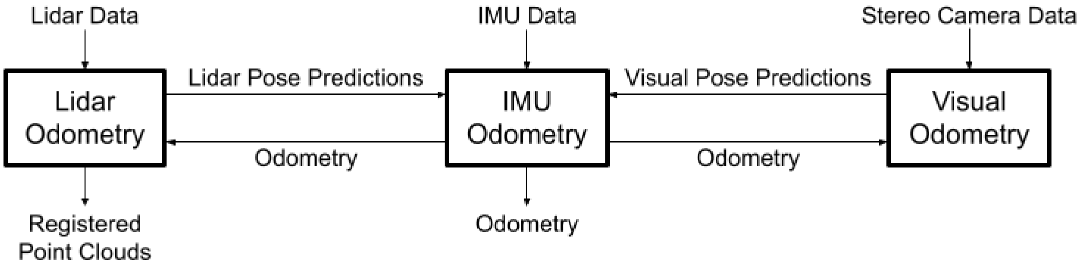
Lidar Data
IMU Data
Stereo Camera Data
Lidar Pose Predictions
Visual Pose Predictions
Lidar Odometry
IMU Odometry
Visual Odometry
Odometry
Odometry
Registered Point Clouds
Odometry

Figure 4.1: Overview of Super Odometry information flow with visual odometry.

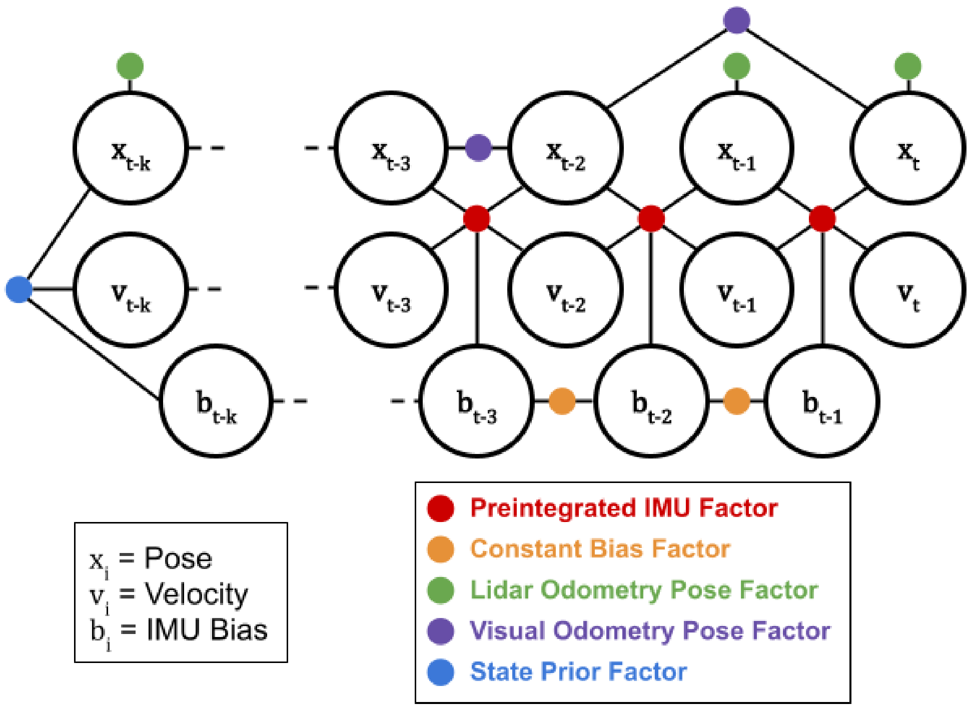
x t-k
v t-k
b t-k
x t-3
x t-2
x t-1
x t
v t-3
v t-2
v t-1
v t
b t-3
b t-2
b t-1
x i = Pose
v i = Velocity
b i = IMU Bias
Preintegrated IMU Factor
Constant Bias Factor
Lidar Odometry Pose Factor
Visual Odometry Pose Factor
State Prior Factor

Figure 4.2: Super Odometry factor graph with binary pose factors from TartanVO.

pose variables. While this is no doubt a better approach in theory, it presents a problem with the current implementation of lidar odometry. This approach would produce a fused odometry result which would not necessarily agree entirely with the pose estimate from lidar odometry alone. The lidar odometry module maintains an internal map of previously-observed geometric features, but the final pose estimate would be "misplaced" in this map as time went on. When the disagreement becomes large enough, lidar odometry can fail.

The final solution is a hybrid of the two discussed above. Pose estimates from lidar odometry are added to the graph as absolute, unary factors, and updates from visual odometry are added as relative, binary factors. An example of this graph structure is shown in figure 4.2. This method forces the fused odometry result to stay relatively close to the estimate from lidar odometry, while still incorporating the information from visual odometry. Note that the actual placement of visual and lidar odometry factors can vary significantly, as there is no synchronization of timing between the two odometry modules. The graph displayed in the figure is simply an example of a possible structure.

4.3 Implementation Notes

The implementation of the factor graph shown in figure 4.2 proved to be far from straightforward. The main reason for this is the lack of synchronization between the lidar and visual odometry modules. With only one source of odometry, pose estimates are always received in chronological order. Updates to the graph are therefore simple and incremental: new variables and factors are added to one end of the graph, and the oldest information is

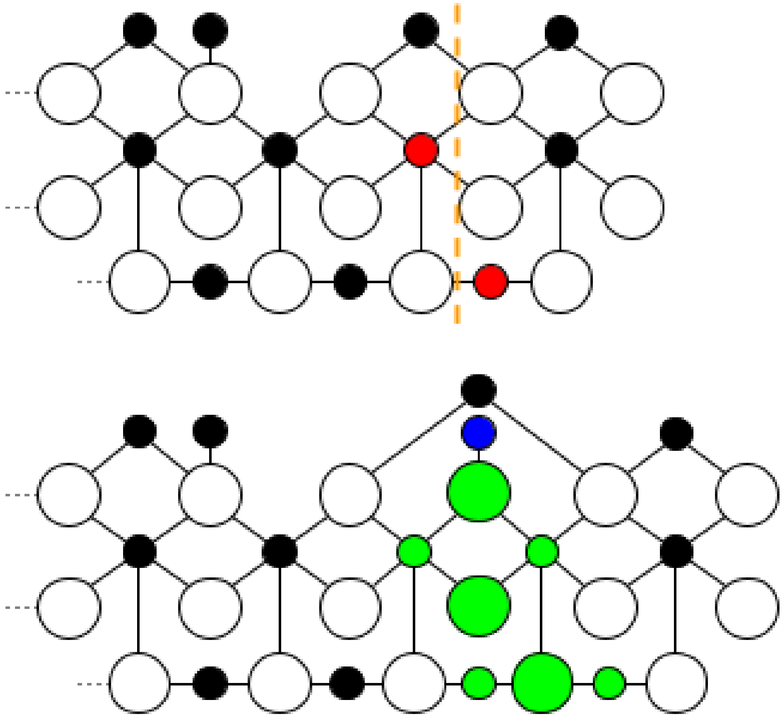

Figure 4.3: Top: an example of a factor graph with both unary and binary pose factors. A unary pose factor will be added to this graph between the third and fourth sets of variables, near the orange dashed line. The red factors will be deleted and replaced. Bottom: the updated factor graph. Variables and factors in green were added so that the new pose factor, in blue, could be added.

marginalized to maintain the fixed-lag smoother. With two unsynchronized sources of pose estimates, there is no guarantee that a received factor will be newer than the most recent factor in the graph. If not, the new factor and its associated variables may have to be stitched into the middle of the graph to keep variables in order. An IMU factor and a bias factor are removed, a new set of variables is added, and new IMU and bias factors are generated to connect these new variables to the rest of the graph. This requires past IMU data to be looked up and reintegrated to produce the new IMU factors. Once this is complete, the new pose factor can be added, and optimization can proceed.

This process is shown in figure 4.3. Labels on factors and variables have been omitted to avoid clutter, but the structure of these graphs is very similar to the graph shown in figure 4.2. The top of figure 4.3 shows the graph, just before a new pose estimate will be incorporated near the orange dashed line. In order to add the new factor, a new set of variables must be introduced. The factors in red are removed, and replaced by the factors and variables in green in the bottom of the figure. After adding the new variables, the new unary pose factor (in blue) can be added to the graph. The process for adding a new binary factor is very similar, though new variables may need to be added on both sides of the new factor.

A Note on Data Structures

As mentioned briefly above, it proved to be very challenging to implement this graph maintenance process. Before incorporating visual odometry, all types of factors were stored together in a single queue. This structure was very logical for maintaining a fixed-lag smoother, since factors were always added to one end of the graph and removed from the other. However, it quickly became unwieldy when adding factors to the middle of the graph. The eventual solution was to separate factors by type (unary pose factors, binary pose factors, IMU factors, bias factors, and priors), and store each subset using an instance of the `map` container in the C++ standard library. This associates keys (timestamps) with values (factors), and uses a binary search tree to sort by key. This approach allows for efficient lookup, insertion, or removal by timestamp, which makes the factor insertion process described above very fast and easy.

Chapter 5

Results

Figures 5.1 and 5.2 compare odometry results to raw GPS ground truth. The lidar-inertial version of Super Odometry was used to generate these plots. Alignment between odometry and GPS trajectories is obtained by fitting the first 20 seconds of the GPS path to odometry. This method allows the observer to examine odometry drift over longer time horizons. The 3.4 km "Course A" shown in figure 5.1 traveled along trails bordered by trees. The 2.75 km "Course B" shown in figure 5.2 traveled through open areas with more sparse vegetation. The larger drift on the latter course was likely due to the significantly higher speed, along with the comparative lack of geometric features for registration. In addition, the loops performed on course A allow for new features to be registered to older parts of the local feature map, creating a form of loop closure and further limiting drift.

Figure 5.3 shows the registered point cloud produced by lidar-inertial Super Odometry operating with a single Velodyne VLP-32C lidar. This lidar is mounted horizontally on the front of the vehicle, and has a somewhat limited field of view. Some cases have been observed where the geometric features observed by this lidar are not sufficient for accurate registration, which can lead to failure of lidar odometry. This was part of the motivation for the multi-lidar version of the system which was discussed in section 3.1.2. Figure 5.4 shows a registered point cloud produced from the same dataset, but taking advantage of three lidars for a more complete field of view. Empirically, this seems to have solved the problem mentioned previously, as it is very likely that at least one of the lidars will detect a reasonable number of features at any given moment.

Figure 5.5 shows odometry results from driving along a straight path in an open area. This is a fairly good odometry solution, as there are no visible jumps or discontinuities in the trajectory. Figure 5.6 shows a top-down view of the vehicle being driven down a narrow trail and around a few sharp curves.

	Course A	**Course B**
Avg. Speed [m/s]	4.81	9.19
Distance [m]	3420	2760
Final Absolute Error [m]	41.61	140.12
Drift Rate	1.22%	5.08%

Table 5.1: Drift statistics of lidar-inertial Super Odometry for two courses.

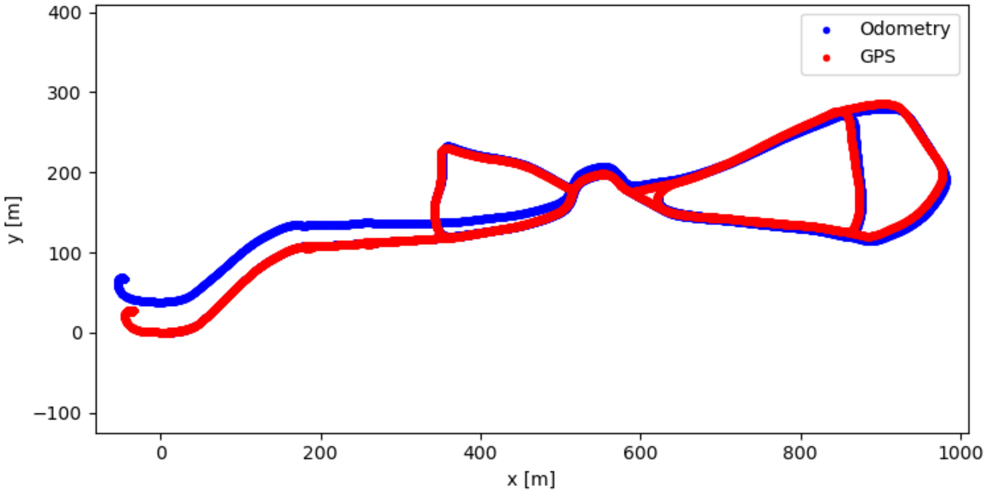

Figure 5.1: Super Odometry result showing drift along 3.4 km of trails (Course A).

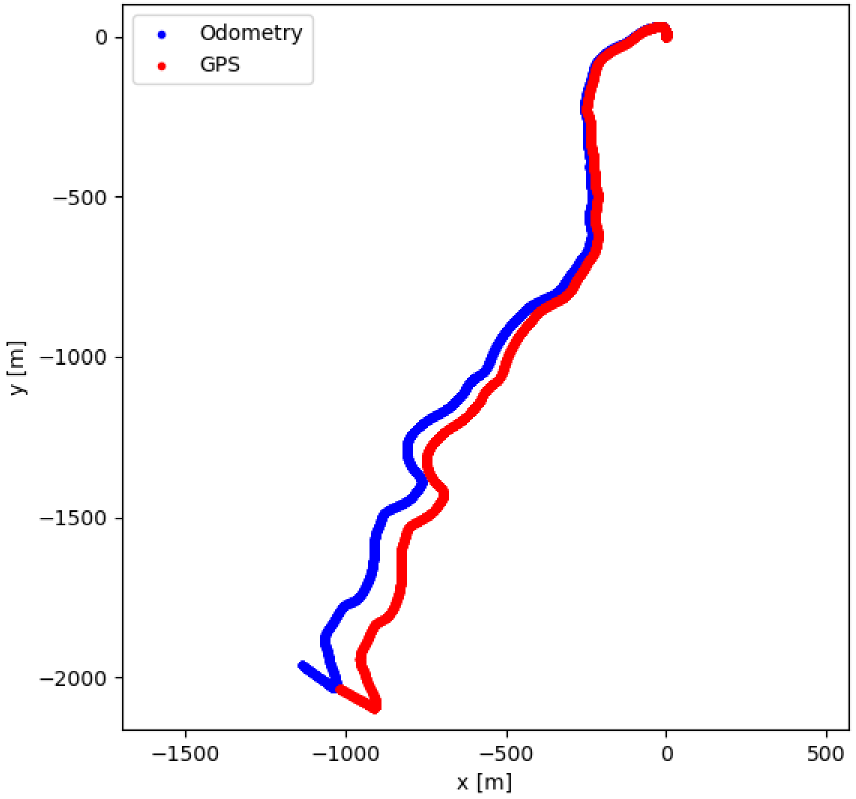

Figure 5.2: Super Odometry result showing drift over 2.75 km in an open area (Course B).

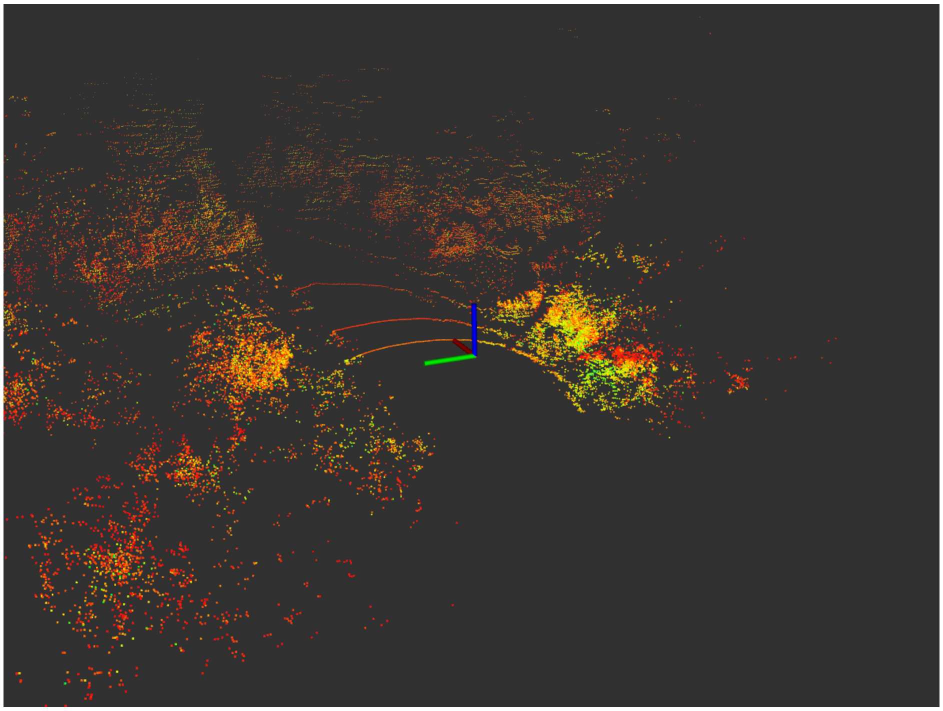

Figure 5.3: Registered point cloud from lidar-inertial Super Odometry with a single lidar.

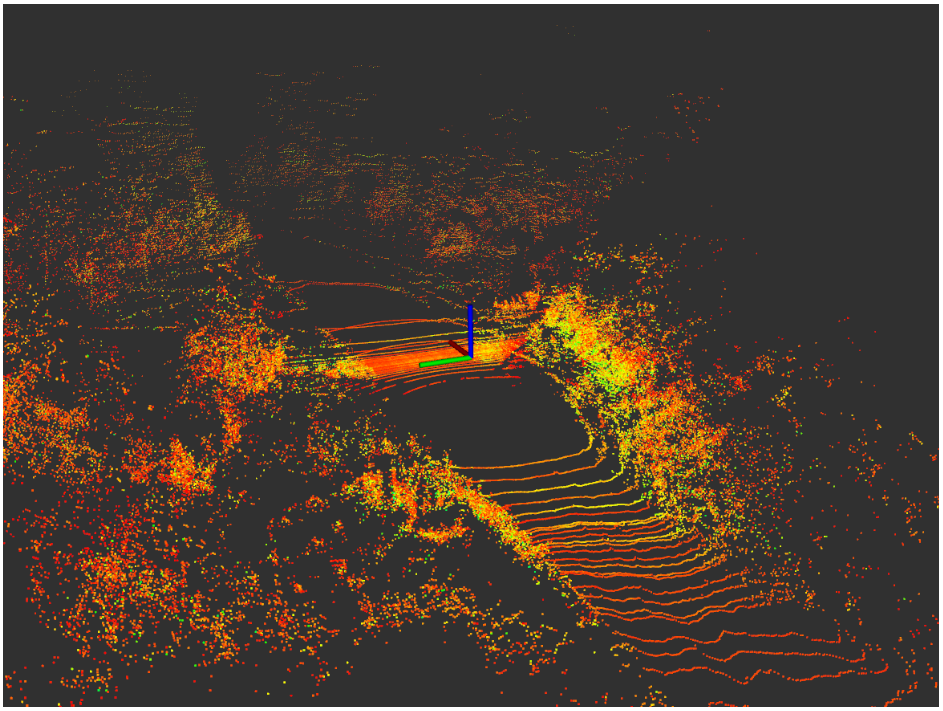

Figure 5.4: Registered point cloud from lidar-inertial Super Odometry with three lidars.

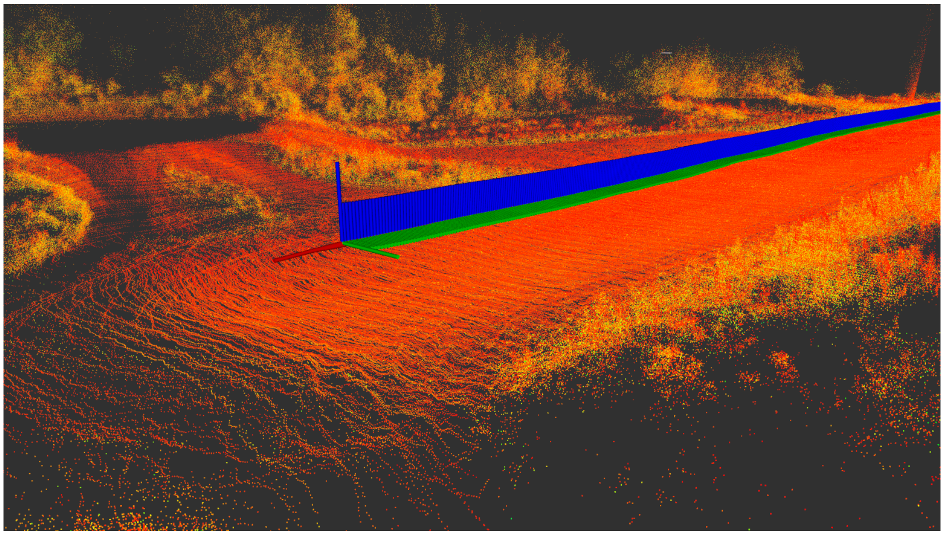

Figure 5.5: Isometric view of odometry and point clouds produced while driving through an open area. The large set of axes on the left shows the most recent pose estimate, and the smaller axes going to the right show previous pose estimates.

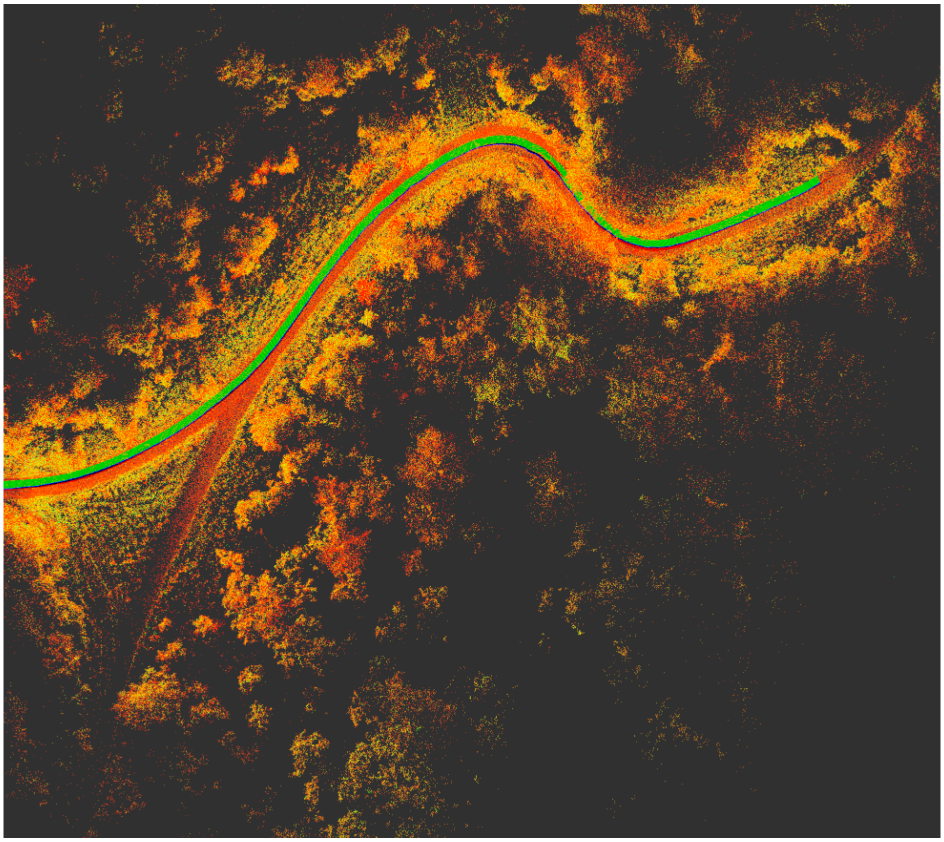

Figure 5.6: Top-down view of odometry and point clouds produced while driving down a narrow trail. The green path shown here is a collection of pose estimates, with the most recent pose near the top right corner of the figure.

Chapter 6

Future Work

6.1 Learning-Based Methods for Uncertainty Estimation

An initial goal of this thesis was to produce a learning-based method to estimate the uncertainty information associated with the pose estimates from visual and lidar odometry. This proved to be a more challenging problem than expected, and was subsequently removed from the scope of this work. This section serves to provide a bit of background on the problem, and describe some of the possible solutions that were considered.

Environmental information such as velocity, acceleration, or the number of observable geometric features could serve as inputs to a small neural network, which would regress to a coefficient α. The value of this coefficient would represent the "trust" that should be put in either the lidar or vision-based odometry results. For example, a value near zero could indicate high trust in visual odometry which would lead to low uncertainty on those pose factors, and a high uncertainty on pose factors from lidar odometry. Conversely, a value near one would have the opposite effect. In theory, these inputs would inform the model of circumstances where the results from one of the odometry methods could degrade, or where one of the methods simply performs better than the other. By inferring an α value and therefore assigning an appropriate covariance to pose estimates, the information contained in the factor graph would more accurately reflect reality.

A key aspect of such an approach is the method used to train the network. The coefficient α is not directly observable, so a means of evaluating a given α value against some other form of ground truth would be necessary. One possible solution would be to compare the final pose estimate—which would be computed using the predicted α value—with a ground truth pose, and deriving a scalar error value from this. This, in turn, requires a ground truth pose estimate, which is not immediately available on the RACER platform. GPS was considered as an option, but was eventually rejected because it would likely contain too much noise to be useful. One would have to evaluate a series of odometry results over a long window of time to overcome the noise in the GPS data, which could defeat the purpose of identifying short-duration degradation in one of the odometry sources. In addition, the GPS data would only provide position information, requiring a different solution for orientation.

Alternatively, IMU information was considered as potential source of ground truth pose.

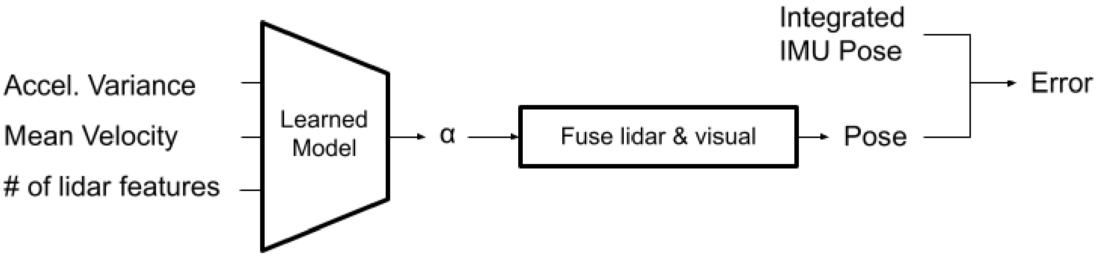

Figure 6.1: Framework considered for learning uncertainty values for pose estimates.

In contrast to GPS, integrated IMU data could be used to provide a reasonable 6 DoF pose estimate over short time intervals. With this concept in mind, the framework shown in figure 6.1 was considered. Although this structure seemed feasible initially, it contains a crippling problem. In order to update the weights in the neural network, one would need the derivative of the final error value with respect to the weights. However, the "fuse lidar & visual" step is not easily differentiable, as this includes the optimization over the factor graph at the heart of Super Odometry. Other potential problems include the fact that inertial information is used both in the fusion step and as ground truth, as well as the lack of synchronization between lidar and visual odometry. For these reasons, this problem remains an open question for future work.

6.2 Correcting for Orientation Drift

One observed drawback of Super Odometry (specifically the lidar-inertial version) is drift in the orientation estimate. Although odometry is initialized at the correct orientation by the gravity-aligned initialization methods described previously, it is not uncommon to see orientation error build up when the vehicle has traveled a significant distance. This issue seems to be a result of the structure of Super Odometry—with the inertial and lidar odometry modules operating separately—and a solution is not immediately clear. The lidar odometry module is nearly independent of the factor graph in the IMU odometry module, only using the final odometry result to initialize the feature registration process. Because of this, orientation drift in the lidar odometry result is essentially guaranteed over longer durations. Because the pose estimates from lidar odometry are added to the factor graph as absolute, unary factors, the inertial information cannot do much to correct for this drift. A true solution to this issue might require some feedback of inertial information into the lidar odometry module to correct the issue at its source. This could have the added benefit of also correcting drift in the registered point clouds, which is a critical component of the problem. In addition, it would likely help to treat pose estimates from lidar odometry as *relative* information, adding them as binary factors in the factor graph. As mentioned in section 4.2, this is not immediately feasible under the current structure.

6.3 Maintaining Performance at Higher Speeds

While this state estimation system has been shown to produce fairly good pose estimates and registered point clouds in most cases, degradation in quality can occur at higher speeds. The primary symptom is misregistration of point clouds, which could indicate two potential problems. First, this could stem from error in the dewarping step, which would cause inaccuracy in the point cloud (and therefore the feature cloud) which is being registered. Such an error would likely be caused by high vibration of the IMU, which could lead to a bad estimates of the vehicle's motion during dewarping. The second potential cause of bad registrations could be derived from the sheer distance traveled between lidar scans. As the speed of the vehicle increases, the displacement that occurs between consecutive scans grows accordingly. This forces the registration step to attempt to reconcile two sets of features are are increasingly different. It is possible that, at a certain speed, accurate registration becomes unlikely for these reasons.

Both of these problems are driven by the relatively low update rate of the lidar. The simplest solution may be to increase the rate at which the lidars are rotating. This is a configurable setting on the Velodyne VLP-32C, and a higher scan frequency could abate some of the issues described above. A more in-depth option would be to change the way that multiple lidars are merged. If two lidars were registered to the map separately—rather than as a single merged feature cloud, as they are currently—that would effectively double the number of registrations that occur, which could help to abate the issue of distance traveled between registrations.

Chapter 7

Conclusion

The primary aim of this work was to produce a reliable and stable state estimate for Carnegie Mellon's team working on the DARPA RACER project. This work was based on Super Odometry [1] and TartanVO [2], fusing separate pose estimates from lidar- and vision-based odometry methods in a central factor graph. Various incremental improvements were made to Super Odometry, and significant effort was put into producing a robust and efficient implementation which could run in real-time. A more accurate point cloud dewarping step (discussed in section 3.1.1) was developed in an attempt to produce an accurate point cloud, even when moving quickly or turning aggressively. This dewarping method was generalized to fuse points from multiple lidars into a single cloud with a common frame of reference. This allowed for a greatly increased field of view, which helped the system maintain a stable state estimate in cases where a single lidar might have been insufficient. The resulting state estimation system has been tested extensively under the RACER program, and has been shown to produce reliable pose estimates and a reasonable point cloud representation of the vehicle's surroundings.

Bibliography

[1] S. Zhao, H. Zhang, P. Wang, L. Nogueira, and S. Scherer, "Super Odometry: IMU-centric LiDAR-visual-inertial estimator for challenging environments," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, (Prague, Czech Republic), pp. 8729–8736, Sept. 2021.

[2] W. Wang, Y. Hu, and S. Scherer, "TartanVO: A generalizable learning-based VO," arXiv, Oct. 2020.

[3] DARPA Public Affairs, "RACER revs up for checkered flag goal of high-speed, off-road autonomy." Available at `https://www.darpa.mil/news-events/2022-01-13`, Jan. 2022.

[4] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, (Shanghai, China), IEEE, May 2011.

[5] S. Agarwal, K. Mierle, and The Ceres Solver Team, "Ceres Solver," Mar. 2022.

[6] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation," in *Proc. Robotics: Science and Systems (RSS)*.

[7] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, (Shanghai, China), pp. 3281–3288, May 2011.

[8] W. Wang, D. Zhu, X. Wang, Y. Hu, Y. Qiu, C. Wang, Y. Hu, A. Kapoor, and S. Scherer, "TartanAir: A dataset to push the limits of visual SLAM," 2020.