Autonomous Flight in Unstructured and Unknown Indoor Environments

by

Abraham Galton Bachrach

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2009

Author . . . .
Department of Electrical Engineering and Computer Science
September 4, 2009

Certified by . . . .
Nicholas Roy
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by . . . .
Terry P. Orlando
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Autonomous Flight in Unstructured and Unknown Indoor Environments

by

Abraham Galton Bachrach

Submitted to the Department of Electrical Engineering and Computer Science
on September 4, 2009, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

This thesis presents the design, implementation, and validation of a system that enables a micro air vehicle to autonomously explore and map unstructured and unknown indoor environments. Such a vehicle would be of considerable use in many real-world applications such as search and rescue, civil engineering inspection, and a host of military tasks where it is dangerous or difficult to send people. While mapping and exploration capabilities are common for ground vehicles today, air vehicles seeking to achieve these capabilities face unique challenges. While there has been recent progress toward sensing, control, and navigation suites for GPS-denied flight, there have been few demonstrations of stable, goal-directed flight in real environments.

The main focus of this research is the development of real-time state estimation techniques that allow our quadrotor helicopter to fly autonomously in indoor, GPS-denied environments. Accomplishing this feat required the development of a large integrated system that brought together many components into a cohesive package. As such, the primary contribution is the development of the complete working system. I show experimental results that illustrate the MAV's ability to navigate accurately in unknown environments, and demonstrate that our algorithms enable the MAV to operate autonomously in a variety of indoor environments.

Thesis Supervisor: Nicholas Roy
Title: Associate Professor of Aeronautics and Astronautics

Acknowledgments

I would like to thank all of the people in my life who have gotten me here. Without your patient support, guidance, friendship and love this thesis, and the work herein would not have been possible. Specifically I would like to thank:

- My advisor, Professor Nicholas Roy for guiding me through this process. The many discussions we have had over the years have helped clarify the concepts and ideas I needed to learn to make this work feasible.
- Ruijie He and Samuel Prentice my conspirators in everything quadrotor related. It has been a lot of fun working with you, and together we have accomplished great things.
- Markus Achtelik for his role in developing the vision system for the quadrotor, and Garrett Hemann for fixing the vehicles when we break them.
- Daniel Gurdan, Jan Stumpf, and the rest of the Ascending Technologies team for creating such impressive vehicles for us.
- Albert Huang for his assistance with the transition to LCM, as well as the many more general algorithm and system design discussions.
- Matt Walter, John Roberts, Rick Cory, Olivier Koch, Tom Kollar, Alex Bahr, and the rest of the people here at CSAIL who have discussed and helped me work through my questions.
- Edwin Olson for providing the reference code for his scan-matching algorithm.
- My wonderful friends in Boston, California, and elsewhere who help distract me from my work when needed to keep me sane. You improve my quality of life.

Last, but certainly not least, Rashmi, Devra, Lela, Bimla, Mom, Dad, and the rest of my family for shaping me into the man I am today with your constant love, support, and encouragement!

Contents

List of Figures

Chapter 1

Introduction

Consider the partially building collapsed shown in figure 1-1. Sending rescue personnel into the building to search for survivors puts them in grave danger. Without knowing what awaits them inside the building, it is very difficult to make good decisions about where it is safe to venture and where to look for survivors. If instead, the building could be searched by a robot, the risks taken by the rescue workers would be greatly diminished. Indeed, there are many situations where it is dangerous and difficult for humans to acquire sensing information and where robots could be of use.

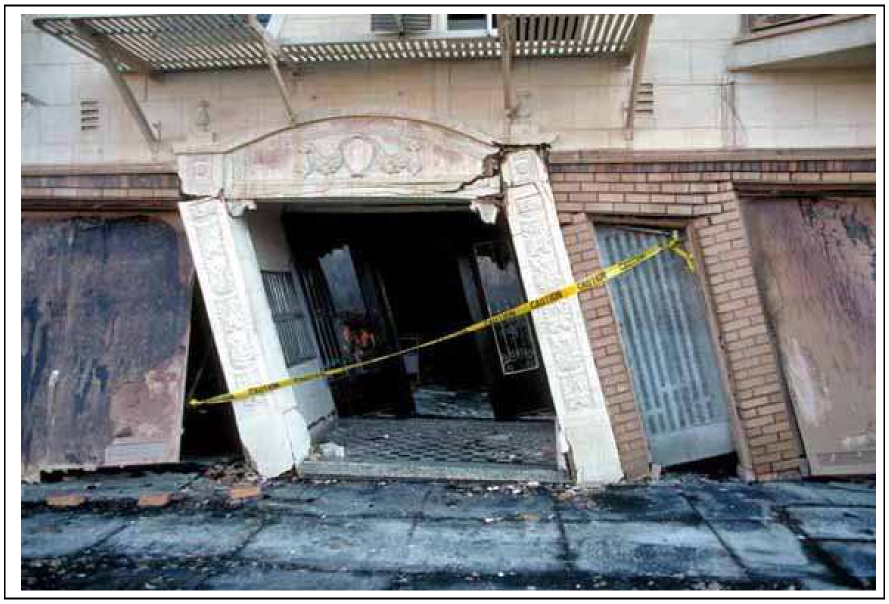


Figure 1-1: A partially collapsed building after an earthquake. [Photo credit: C.E. Meyer, U.S. Geological Survey]

While the utility of robots performing such sensing tasks may be obvious, creating the robots is certainly not. Operating within a partially collapsed building, or other similar environments requires a robot to be able to traverse cluttered, obstacle strewn terrain. Over the years, researchers have tackled these problems and designed a number of ground robot systems, such as the ones shown in figure 1-2 capable of traversing rough terrain. Despite the progress toward this goal, it is still an active area of research and no matter how far the field advances, there will always be some terrain which a ground robot is simply not physically capable of climbing over. Many researchers have therefore proposed the use of Micro Air Vehicles (MAVs) as an alternative robotic platform for rescue tasks and a host of other applications.

Figure 1-2: Two ground robots designed for traversing rough terrain. [Photo credit: (a)DARPA Learning Locomotion Project at MIT, (b) NIST]

Indeed, MAVs are already being used in several military and civilian domains, including surveillance operations, weather observation, disaster relief coordination, and civil engineering inspections. Enabled by the combination of GPS and MEMs inertial sensors, researchers have been able to develop MAVs that display an impressive array of capabilities in outdoor environments without human intervention.

Unfortunately, most indoor environments and many parts of the urban canyon remain without access to external positioning systems such as GPS. Autonomous MAVs today are thus very limited in their ability to operate in these areas. Traditionally, unmanned vehicles operating in GPS-denied environments can rely on dead reckoning for localization, but these measurements drift over time. Alternatively, with onboard environmental sensors, simultaneous localization and mapping (SLAM) algorithms build a map of the environment around the vehicle from sensor data while simultaneously using the data to estimate the vehicle's position. Although there have been significant advances in developing accurate, drift-free SLAM algorithms in large-scale environments, these algorithms have focused almost exclusively on ground or underwater vehicles. In contrast, attempts to achieve the same results with MAVs have not been as successful due to a combination of limited payloads for sensing and computation, coupled with the fast and unstable dynamics of air vehicles. While MAV platforms present the promise of allowing researchers to simply fly over rough and challenging terrain, MAVs have their own host of challenges which must be tackled before this promise can be realized.

1.1 Key Challenges

In the ground robotics domain, combining wheel odometry with sensors such as laser range-finders, sonars, or cameras in a probabilistic SLAM framework has proven very successful [92]. Many algorithms exist that accurately localize ground robots in large-scale environments; however, experiments with these algorithms are usually performed with stable, slow moving robots such as the ones shown in figure 1-3, which cannot handle even moderately rough terrain.

Unfortunately, mounting equivalent sensors onto a MAV and using an existing SLAM algorithms does not result in the same success. MAVs face a number of unique challenges that make developing algorithms for them far more difficult than their indoor ground robot counterparts. The requirements and assumptions that can be made with flying robots are sufficiently different that they must be explicitly reasoned about and managed differently.

Limited Sensing Payload MAVs have a maximum amount of vertical thrust that they can generate to remain airborne, which severely limits the amount of payload available for sensing and computation compared to similar sized ground vehicles. This weight limitation eliminates popular sensors such as SICK laser scanners, large-aperture cameras and

Figure 1-3: Examples of robots commonly used for SLAM research. [Photo credit: Cyrill Stachniss]

high-fidelity IMUs. Instead, indoor air robots must rely on lightweight Hokuyo laser scanners, micro cameras and lower-quality MEMS-based IMUs, which generally have limited ranges, fields-of-view and are noisier compared to their ground equivalents.

Limited Onboard Computation Despite the advances within the community, SLAM algorithms continue to be computationally demanding even for powerful desktop computers and are therefore not usable on today's small embedded computer systems that might be mounted onboard MAVs. The computation can be offloaded to a powerful ground-station by transmitting the sensor data wirelessly; however, communication bandwidth then becomes a bottleneck that constrains sensor options. For example, camera data must be compressed with lossy algorithms before it can be transmitted over wireless links, which adds noise and delay to the measurements. The delay is in addition to the time taken to transmit the data over the wireless link. The noise from the lossy compression artifacts can be particularly damaging for feature detectors that look for high frequency information such as corners in an image. Additionally, while the delay can often be ignored for slow moving, passively stable ground robots, MAVs have fast and unstable dynamics, making control under large sensor delay conditions impossible.

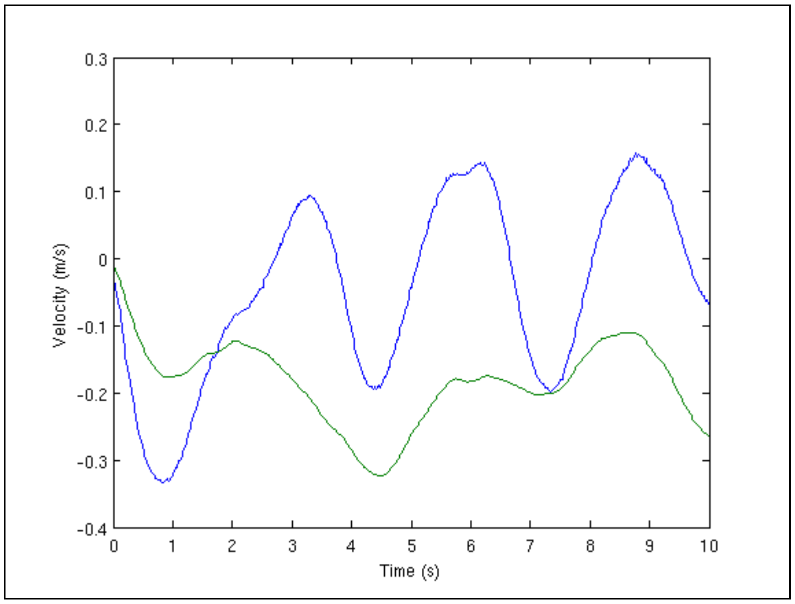


Figure 1-4: Ground truth velocities (blue) compared with integrated acceleration (green). In just 10 seconds, the velocity estimate diverged by over $.25m/s$. Position estimates would diverge commensurately faster.

Indirect Relative Position Estimates Air vehicles do not maintain physical contact with their surroundings and are therefore unable to measure odometry directly, which most SLAM algorithms require to initialize the estimates of the vehicle's motion between time steps. Although one can compute the relative motion by double-integrating accelerations, lightweight MEMs IMUs are often subject to unsteady biases that result in large drift rates, as shown in figure 1-4. We must therefore recover the vehicle's relative motion indirectly using exteroceptive sensors, and computing the vehicle's motion relative to reference points in the environment.

Fast Dynamics MAVs have fast dynamics, which results in a host of sensing, estimation, control and planning implications for the vehicle. When confronted with noisy sensor measurements, filtering techniques such as Kalman Filters are often used to obtain better estimates of the true vehicle state. However, the averaging process implicit in the these filters mean that multiple measurements must be observed before the estimate of the underlying state will change. Smoothing the data generates a cleaner signal, but adds delay to the state estimates. While delays may have insignificant effects on vehicles with slow dynamics, the effects are amplified by the MAV's fast dynamics. This problem is illustrated in figure 1-5, where we compare the normal hover accuracy to when state estimates are delayed by 150ms. While our vehicle is normally able to hover with an RMS error of $6cm$, with the delay, the error increases to $18cm$.

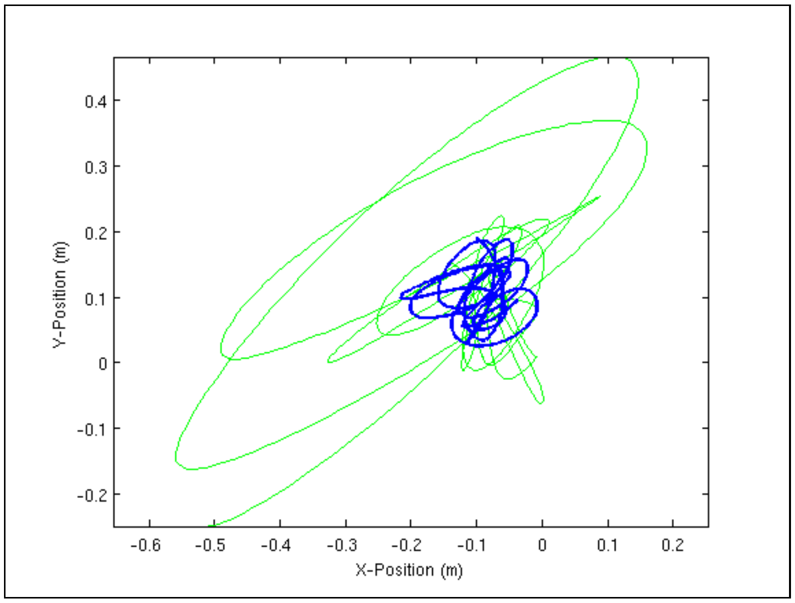


Figure 1-5: Comparison of the hover accuracy using the state estimates from our system without additional delay (blue), and the accuracy with 150ms of delay artificially imposed (green).

Need to Estimate Velocity In addition, as will be discussed further in Section 3.3, MAVs such as the quadrotor that we use are well-modeled as a simple 2^{nd}-order dynamic system with no damping. The underdamped nature of the dynamics model implies that simple proportional control techniques are insufficient to stabilize the vehicle, since any delay in the system will result in unstable oscillations, an effect that we have observed experimentally. We must therefore add damping to the system through the feedback controller, which emphasizes the importance of obtaining accurate and timely state estimates for both position and velocity. Traditionally, most SLAM algorithms for ground robots completely ignore the velocity states.

Constant Motion Unlike ground vehicles, a MAV cannot simply stop and perform more sensing when when its state estimates have large uncertainties. Instead, the vehicle is likely to be unable to estimate its velocity accurately, and as a result, it may pick up speed or oscillate, degrading the sensor measurements further. Therefore, planning algorithms for air vehicles must not only be biased towards paths with smooth motions, but must also explicitly reason about uncertainty in path planning, as demonstrated in [41]; motivating our exploration strategy in section 3.5.

3D Motion Finally, MAVs operate in a truly $3D$ environment since they can hover at different heights. While it is reasonable for a ground robot to focus on estimating a $2D$ map of the environment, for air vehicles, the $2D$ cross section of a $3D$ environment can change drastically with height and attitude, as obstacles suddenly appear or disappear. However, if we explicitly reason about the effects of changes due to the $3D$ structure of the environment, we have found that a $2D$ representation of the environment is surprisingly useful for MAV flight.

1.2 Problem Statement

In the research presented in this thesis, we sought to tackle the the problems described above and develop a system that integrates sensing, planning, and control to enable a MAV to autonomously explore indoor environments. We seek to do this using only onboard sensing and without prior knowledge of the environment.

1.3 Related Work

In recent years, the development of autonomous flying robots has been an area of increasing research interest. This research has produced a number of systems with a wide range of capabilities when operating in outdoor environments. For example vehicles have been developed that can perform high-speed flight through cluttered environments [84], or even acrobatics [20]. Other researchers have developed systems capable of autonomous landing, terrain mapping [90], and a host of high level capabilities such as coordinated tracking and planning of ground vehicles [12], and multi-vehicle coordination [32, 93, 17]. While these are all challenging research areas in their own right, and pieces of the work (such as the modeling and control techniques) carry over to the development of vehicles operating in indoors, these systems rely on external systems such as GPS, or external cameras [63] for localization. In our work, we focus on flying robots that are able to operate autonomously while carrying all sensors used for localization, control and navigation onboard. This is in contrast to approaches taken by other researchers [45, 43] who have flown indoors using position information from motion capture systems, or external cameras [8, 9].

Outdoor Visual Control While outdoor vehicles can usually rely on GPS, there are many situation where it would be unsafe for a vehicle to rely on it, since signal can be lost due to multipath fading, satellites being occluded by buildings and foliage, or even intentional jamming. In response to these concerns, a number of researchers have developed systems that rely on vision for control of the vehicle. The capabilities of these systems include visual servoing relative to a designated target [64], landing on a moving target [80], and even navigation through urban canyons [47]. While the systems developed by these researchers share many of the challenges faced by indoor MAVs, they operate on vehicles that are orders of magnitude larger, such as the one shown in figure 1-6, with much greater sensing and computation payloads.

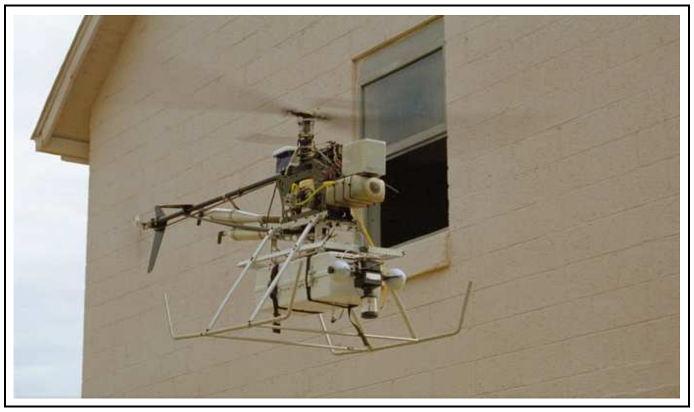


Figure 1-6: The USC AVATAR helicopter, built around a Bergen Industrial Twin RC helicopter, which is a common helicopter platform for outdoor experiments. [Photo credit: Dr. Stefan Hrabar and the USC Robotic Embedded Systems Laboratory (`http://robotics.usc.edu/resl`)]

Indoor Obstacle Avoidance Using platforms that are of a similar scale to the ones targeted in this thesis, several researchers [75, 14, 62] use a small number of ultrasound or infrared sensors to perform altitude control and basic obstacle avoidance in indoor environments. While their MAVs are able to hover autonomously, they do not achieve any sort of goal directed flight that would enable the systems to be controlled at a high level such that they could be used for higher level applications.

Known Structure Instead of using low resolution sonar and infrared sensors, several authors have attempted to fly MAVs autonomously indoors using monocular camera sensors. To enable the vision processing to be tractable, they make very strong (and brittle) assumptions about the environment. For example, Tournier et al. [95] performed visual servoing over known Moire patterns to extract the full 6 *dof* state of the vehicle for control; and Kemp [54] fit lines in the camera images to the edges of a $3D$ model of an office environment with known structure. Reducing the prior knowledge slightly, Johnson [49] detects lines in a hallway, and used the assumption of a straight hallway to back out the vehicle pose. Similarly, Celik et al presented their MVCSLAM system in [19], which tracks corner features along the floor of a hallway. While impressive, it is unclear how their work could be extended to other environments. Their applicability is therefore constrained to environments with specific features, and thus may not work as well for general navigation in GPS-denied environments.

Using a $2D$ laser scanner instead of a camera, prior work done in our group [41] presented a planning algorithm for a quadrotor helicopter that is able to navigate autonomously within an indoor environment for which there is a known map. Recently, [10, 35] designed quadrotor configurations that were similar to the one presented in [41]. Grzonka et al. and Angeletti et al. [10] scan-matched successive laser scans to hover their quadrotor helicopter, while [35, 36] used particle filter methods to globally localize their MAV. However, none of these papers presented experimental results demonstrating the ability to stabilize all 6 degrees of freedom of the MAV using the onboard sensors, and all made use of prior maps, an assumption that is relaxed in this thesis.

Indoor SLAM Finally, perhaps the closest work to ours was that of Ahrens [7], where monocular vision SLAM was used to stabilize a quadrotor. Extracted corner features were fed into an extended Kalman filter based vision-SLAM framework, building a low-resolution $3D$ map sufficient for localization and planning. Unfortunately, an external motion capture system was used to simulate inertial sensor readings, instead of using an on-board IMU. As such, their system was constrained to the motion capture volume where they had access to the high quality simulated IMU. It remains to be seen whether the work can be extended to use lower quality acceleration estimates from a more realistic MAV-scale IMU.

Adopting a slightly different approach, Steder et al [86] mounted a downward-pointing camera on a blimp to create visual maps of the environment floor. While interesting algorithmically, this work does not tackle any of the challenges due to the fast dynamics described in section 1.1.

Where many of the above approaches for indoor flight fall short is that they did not consider the requirements for stabilizing the MAV both locally and in larger scale environments in a coherent system. The previous work either focused on local hovering and obstacle avoidance in known, constrained environments, or attempted to tackle the full SLAM problem directly, without stabilizing the vehicle using local state estimation methods. SLAM processes are generally too slow to close the loop and control an unsteady MAV, resulting in systems that work well in simulation, however are unworkable when applied on real hardware. In our work, we developed a multi-layer sensing and control hierarchy which tackles both of these challenges in a coherent system.

1.4 Contributions

In this thesis, I present the design, implementation, and validation of a system for localizing and controlling the quadrotor helicopter shown flying in figure 1-7, such that it is capable of autonomous goal directed flight in unstructured indoor environments. As such, the primary contribution is the development of the working system. While the system builds upon existing work from the robotics community, many of the individual components required

Figure 1-7: A photo of our vehicle flying autonomously in an unstructured indoor environment

adaptation to be used on MAVs. More specifically, the contributions of this thesis are:

1. Development of a fully autonomous quadrotor that relies only on onboard sensors for stable control, without requiring prior information (maps) about the environment.

2. A high-speed laser scan-matching algorithm that allows successive laser scans to be compared in real-time to provide accurate velocity and relative position information.

3. An Extended Kalman Filter data fusion module, and algorithm for tuning it that provides accurate real-time estimates of the MAV position and velocity

4. A modified SLAM algorithm that handles the $3D$ environment structure in a $2D$ map

5. A framework for performing dense reconstruction and mapping of the full $3D$ environment around the vehicle

6. A visual object tracking system that allows the vehicle to follow a target designated in a camera image.

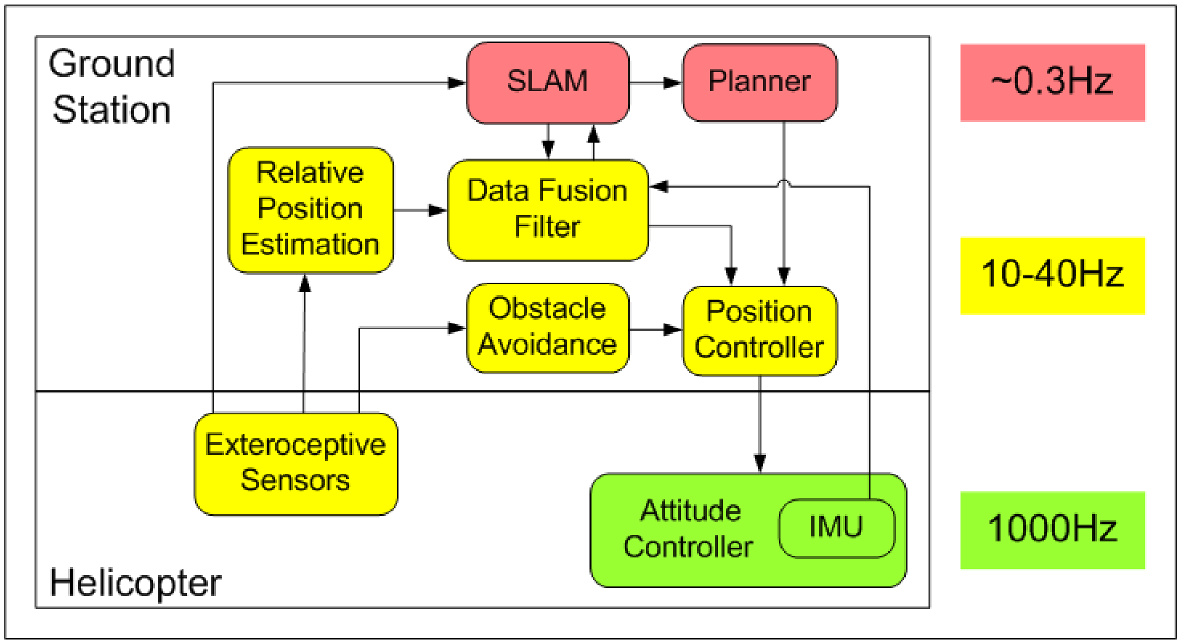


Figure 1-8: Schematic of our hierarchical sensing, control and planning system. At the base level, the onboard IMU and controller (green) creates a tight feedback loop to stabilize the MAV's pitch and roll. The yellow modules make up the real-time sensing and control loop that stabilizes the MAV's pose at the local level and avoids obstacles. Finally, the red modules provide the high-level mapping and planning functionalities.

1.5 System Overview

To compute the high-precision, low delay state estimates required for indoor flight, we designed the 3-level sensing and control hierarchy, shown in figure 1-8, distinguishing processes based on the real-time requirements of their respective outputs. The first two layers run in real-time, and are responsible for stabilizing the vehicle and performing low level obstacle avoidance. The third layer is responsible for creating a consistent global map of the world, as well as planning and executing high level actions.

At the base level, the onboard IMU and processor creates a very tight feedback loop to stabilize the MAV's pitch and roll, operating at $1000Hz$. At the next level, fast, high-resolution relative position estimation algorithms, described in chapter 2, estimate the vehicle's motion, while an Extended Kalman Filter (EKF) fuses the estimates with the IMU outputs to provide accurate, high frequency state estimates. These estimates enable the LQR-based feedback controller to hover the MAV stably in small, local environments. In addition, a simple obstacle avoidance module ensures that the MAV maintains a minimum distance from observed obstacles.

At the top layer, a SLAM algorithm uses the EKF state estimates and incoming laser scans to create a global map, ensuring globally consistent state estimates by performing loop closures. Since the SLAM algorithm takes 1-2 seconds to incorporate incoming scans, it is not part of the real-time feedback control loops at the lower levels. Instead, it provides delayed correction signals to the EKF, ensuring that our real-time state estimates remain globally consistent. Finally, a planning and exploration module enables the vehicle to plan paths within the map generated by the SLAM module, and guide the vehicle towards unexplored regions.

1.6 Outline

In the chapters 2 and 3, I describe the components of the system that enable flight in unconstrained indoor environments. Chapter 2 covers the algorithms for obtaining relative position estimates using either high-speed laser scan-matching or stereo visual odometry. Chapter 3 describes how these estimates are used in the complete system, along with the details of the system components.

After describing the complete system, I describe the hardware we use, and present experimental results validating our design and demonstrating the capabilities of our system in chapter 4.

In chapter 5 I present a framework for performing dense mapping of the $3D$ environment around the vehicle, which would enable generating motion plans in $3D$. Finally, in chapter 6 I present a vision based object tracking system that allows the vehicle to perform high level tasks such as following a person before presenting future work and concluding remarks in chapter 7.

Chapter 2

Relative Position Estimation

In this chapter I describe the algorithms used for estimating the MAV's relative position in real-time. Both laser and stereo-vision based solutions are presented and compared. The high quality relative position estimates provided by these algorithms are the key enabling technology for indoor flight.

The stereo-vision based visual odometry solution was developed in collaboration with Markus Achtelik [6].

2.1 Introduction

MAVs have no direct way to measure their motion, and must therefore rely on sophisticated algorithms to extract synthetic proxies for the wheel encoder based odometry available on ground robots from other sensors. While one may be tempted to double-integrate acceleration measurements from inertial sensors to obtain relative position estimates, the drift rates of small lightweight MEMs IMUs are prohibitively high. Instead one must rely on exteroceptive sensors, matching incoming measurements with one another to back out the vehicle's relative motion. This process can be performed on both laser scans and camera images, each having distinct advantages and disadvantages in terms of their computational requirements, accuracy, and failure modes.

While air vehicles do not have the luxury of using wheel odometry to measure relative position, many ground robots have faced similar challenges, since in many situations the estimates from wheel odometry can be quite poor, such as when a robot is traversing rough terrain. As a result, there has been considerable work on developing relative position estimation algorithms for ground robots. Researchers have often found that the performance of both scan-matching and visual odometry greatly outperforms wheel odometry [70, 46]. Although the algorithms have largely been developed for ground robots, they can be adapted for use on MAVs, but with the additional challenging requirements of both high resolution matching and fast real-time performance. The high resolution matching is particularly important due to the need to estimate the velocity of the MAV for control purposes. To obtain these velocity estimates, we must differentiate the computed relative position estimates. So while a positional error of a few centimeters may be insignificant for a ground robot, when divided by the time between scans, a few centimeter error in position will result in a large error in velocity. In addition, since MAVs operate in the full 3D environment, we must ensure that the algorithms are robust to motion in all 6 degrees of freedom.

The relative position estimation algorithms can generally be broken down into two subroutines:

1. **Correspondence:** Find matches between "features" in the measurements
2. **Motion Extraction:** Given sets of corresponding features, determine the optimal rigid body transform between them.

As we shall see, the existing algorithms take different approaches to each of these subroutines, with different robustness, accuracy, and computational complexity properties.

In addition, different types of sensors have unique characteristics that lead to varying levels of effectiveness in different environments. Laser range-finders operate by emitting a beam of laser light, and measuring the time until the beam is reflected back onto a photo sensor. This process provides a measurement of the distance to the nearest obstacle in the direction of the laser beam. By sweeping the laser beam in a circle, and taking successive point measurements at fixed intervals, the sensor is able to generate a "scan" of the environment that contains the range to the nearest obstacle at a fixed set of bearings. When converted from this polar-coordinate form to Cartesian coordinates we obtain a set of points such as the ones shown in Figure 2-1(a). Since laser range finders provide a set of distances, laser scans can only be matched when the environment has unique physical structure or shape. As a result, the matching process can fail around homogeneous building structures such as long corridors. In addition, since the sensors only generate $2D$ slices of the environments, they cannot make use of structure outside the sensing plane.

In contrast, camera sensors, which measure the intensity of light falling onto a $2D$ sensor plane, can make use of information from the full $3D$ environment around the vehicle. However, camera sensors only measure the intensity of the light, and therefore do not provide direct information about the underlying $3D$ structure that generated the image. To be able to extract that information from image data the environment must contain unique visual features, and requires sophisticated image processing. In general, camera sensors have more limited angular field-of-views and are computationally intensive to work with.

Different exteroceptive sensors are therefore better suited for autonomous MAV operation under different environmental conditions. However, since the laser scanner and cameras rely on different environmental features, they should have complementary failure modes. As a result, integrating both sensors onto a single MAV platform will enable autonomous navigation in a wide range of generic, unstructured indoor environments.

2.2 Laser Scan-Matching

Laser scan-matching algorithms must solve the following problem: given two overlapping laser range scans $S_t \in \Re^{2\times n}$ and $S_{t-1} \in \Re^{2\times n}$, find the optimal rigid body transform $\Delta \in SO(3) = [R, t]$ that aligns the current laser scan with the previous scan such that applying the transform Δ to S_{t-1}, denoted $\Delta \otimes S_{t-1}$, results in a scan that is close to S_t. To find the best alignment, one needs a method for scoring candidate transforms based on how well they align to past scans. The first challenge in doing this is that laser scanners provide individual point measurements of locations in the environment. Successive scans will generally not measure the same points in the environment due to the motion of the vehicle. Each scan-matching algorithm must therefore find a way to overcome this issue to find correspondences. This is usually done in one of 3 ways:

1. **Point-to-Point:** The individual points from the current scan are matched directly to one (or multiple) points in the previous scan, such as in the Iterative Closest Point (ICP) [101] algorithm described below.

2. **Feature-to-Feature:** Points are grouped into higher level features such as corners and lines that are then matched, such as in the HAYAI algorithm [58]. Since the features can often be accurately corresponded directly, the resulting motion estimate can be computed in closed form, without needing the iterative refinement of the correspondences used in ICP.

3. **No-Correspondences:** The points from previous scans are used to create a likelihood map, and the current scan is matched to this map by searching for the pose which gives the scan a maximum likelihood with respect to the map. This is the approach taken by, Vasco [39, 2], Olson [70], and our scan-matcher. One of the benefits of this approach is that it does not require explicit correspondences to be computed.

When considering the algorithm to use on a MAV, robustness to outliers is particularly important. The laser scanner measures ranges in a 2D plane, while the vehicle moves in the full 3D environment. Motion out of the plane of the laser can result in portions of the laser scan changing dramatically. As a result, while the scan-matching algorithms for ground robots must only worry about errors due to sensor noise, which is generally fairly low, our algorithm must be very robust to regions of the scan changing due to 3D effects. This requirement essentially precludes the use of the feature based approaches since they are very susceptible to incorrect correspondences between features.

2.2.1 Iterative Closest Point

The iterative closest point algorithm [101] is one of the simplest and most commonly used algorithms for laser scan-matching. It is an iterative algorithm which alternates between finding correspondences between individual points and their nearest neighbor, and finding the optimal rigid body transform that minimizes the Euclidean distance between the points, as shown in algorithm 1. The optimal rigid body transform (in a least-squares sense) between two sets of corresponded points can be computed in closed form [97], which is

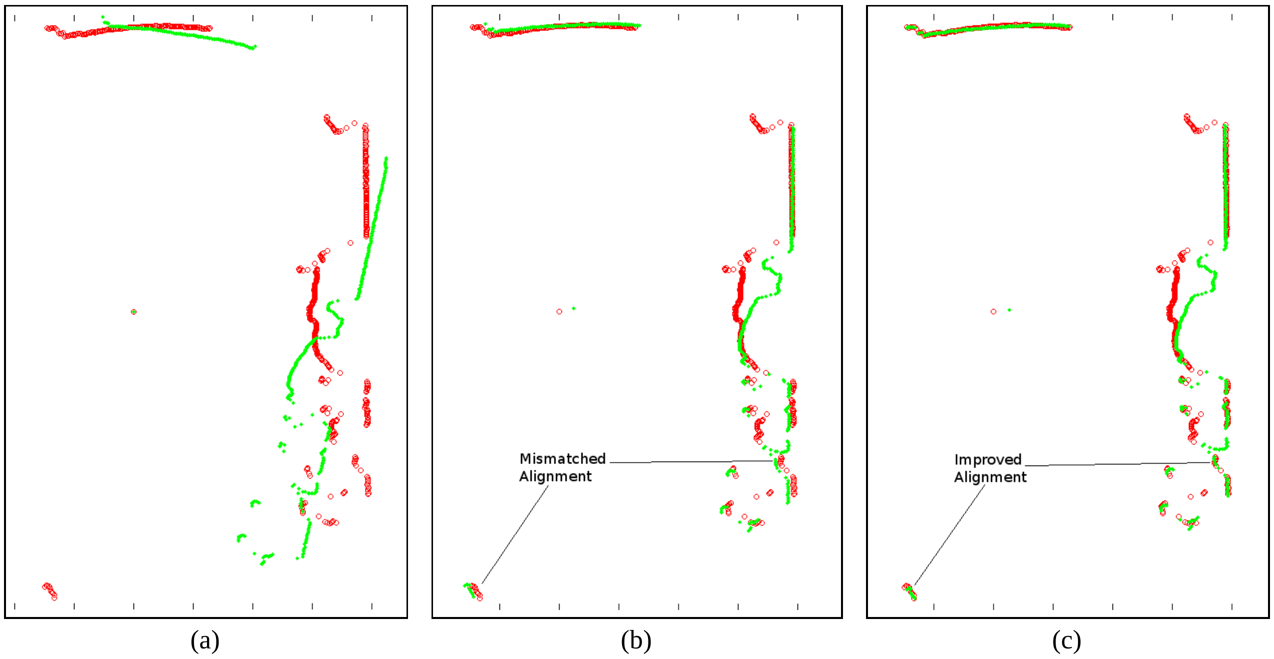


Figure 2-1: (a) The original set of points for 2 laser scans, showing a region with significant differences due to 3D effects. (b)The alignment computed by ICP. Notice the misalignment due to poor correspondences in the non-matching region. (c) The alignment computed by our algorithm, which robustly matches the two scans.

attractive due to its computational efficiency. While the rigid body transform for a given set of corresponding points is computed in closed form, the algorithm as a whole does not necessarily find the global optimum, since it may be susceptible to local minima when computing the point correspondences.

While this basic form of the ICP algorithm is extremely simple and fairly efficient, it suffers from robustness issues in the presence of regions of the scans that do not match, such as the scans in figure 2-1(a). The region in the bottom right of the figure has considerable differences between the two scans due to out of plane motion. Since the ICP algorithm finds correspondences for these points despite the fact that they have no match, it skews the computed transform, resulting in the slight rotation between the scans in figure 2-1(b). Many researchers have proposed variants of this basic ICP algorithm, which use a different distance function, or add an outlier rejection step which improves the matching robustness and/or efficiency [78, 77, 66]. However these variants add significant complexity without solving the fundamental problem: corresponding points which are not measurements of the

Algorithm 1 The Iterative Closest Point algorithm

```
Require: $S_1$ and $S_2$    (the scans to be matched)
Require: $\Delta$    (initial guess of the transformation)
  while $\Delta$ not converged do
    $\hat{S}_2 = \Delta \otimes S_2$    (project $S_2$ using current transform)
    for $\mathbf{x}_i^1 \in S_1$ do
      $\mathbf{y}_i = \underset{\mathbf{x}_i^2 \in \hat{S}_2}{\operatorname{argmin}} ||\mathbf{x}_i^1 - \mathbf{x}_i^2||_2$
    end for
    $\Delta = [R, t] = \underset{[R,t]}{\operatorname{argmin}} \sum_{i=1}^{N} ||R\mathbf{y}_i + t - \mathbf{x}_i^1||_2$
  end while
```

same point in the environment.

2.2.2 Map Based Probabilistic Scan-Matching

As an alternative to computing the correspondences explicitly, a challenging and error prone process, one can create an occupancy grid map M from previous scans [91], and then match the new scan against that map. Each cell in the map stores the likelihood of the ith laser return being measured at the point x_i as

$$P(x_i|M) \tag{2.1}$$

This map then allows us to compute the likelihood for an entire scan by computing the likelihood that all laser readings fall where they do. Assuming that each of the point measurements in a laser scan are independent, the likelihood for an entire scan can be computed as

$$P(S|M) = \prod_{i=1}^{N} P(x_i|M) \tag{2.2}$$

By searching over candidate rigid body transforms $\Delta \in SO(3)$ to find the one that maximizes the likelihood of the laser reading, we can then find the optimal Δ^* which provides the best alignment of the incoming laser scan:

$$\Delta^* = \underset{\Delta}{\operatorname{argmax}} P(\Delta \otimes S|M) \tag{2.3}$$

where $\Delta \otimes S$ is the set of laser points S transformed by the rigid body transform Δ

This is the approach used by the Vasco scan-matcher in the Carmen robotics toolkit [39, 2], and the scan-matcher developed by Olson [70, 69], upon which our work is based. While similar in concept, the algorithms differ in their methods for constructing and searching the likelihood map.

In Vasco the map is constructed by storing the number of times each cell is hit by a laser measurement and then integrating over small errors with Gaussian smoothing. This smoothing captures the uncertainty present in the laser measurements. There is noise in both the range and bearing of each laser measurement, which means that points near a cell that is hit should also have an increased likelihood of generating a laser return. The optimal scan alignment with respect to the map is then computed by performing a greedy hill-climbing process. Starting from an initial guess of the correct transform, Vasco successively tests new transforms around the current best, and keeps modifications that increase the likelihood of the resulting transformed scan as shown in algorithm 2.

Algorithm 2 Vasco Hill Climbing Algorithm

```
Require: $S$      (scan to be matched)
Require: $M$      (the likelihood map)
Require: $\Delta$      (initial guess of the transformation)
  $L \leftarrow P(\Delta \otimes S|M)$ (evaluate likelihood of initial guess)
  while $\Delta$ not converged do
    for $\hat{\Delta} = \Delta + \delta \in \{Forward, Back, Left, Right, TurnLeft, TurnRight\}$ do
      if $P(\hat{\Delta} \otimes S|M) > L$ then
        $L \leftarrow P(\hat{\Delta} \otimes S|M)$
        $\Delta \leftarrow \hat{\Delta}$
      end if
    end for
  end while
```

While this scan-matching method has proven quite successful, and is an improvement over ICP, it still has two flaws which are corrected in Olson's approach. First, since laser scanners have a limited angular resolution, readings far from the sensor will be spaced far apart. As a result, if a subsequent measurement at time $t + 1$ falls in between two readings taken at time t, the later measurement will appear to be low likelihood despite the fact that it probably reflected off the same object. Second, and more worrisome for use on

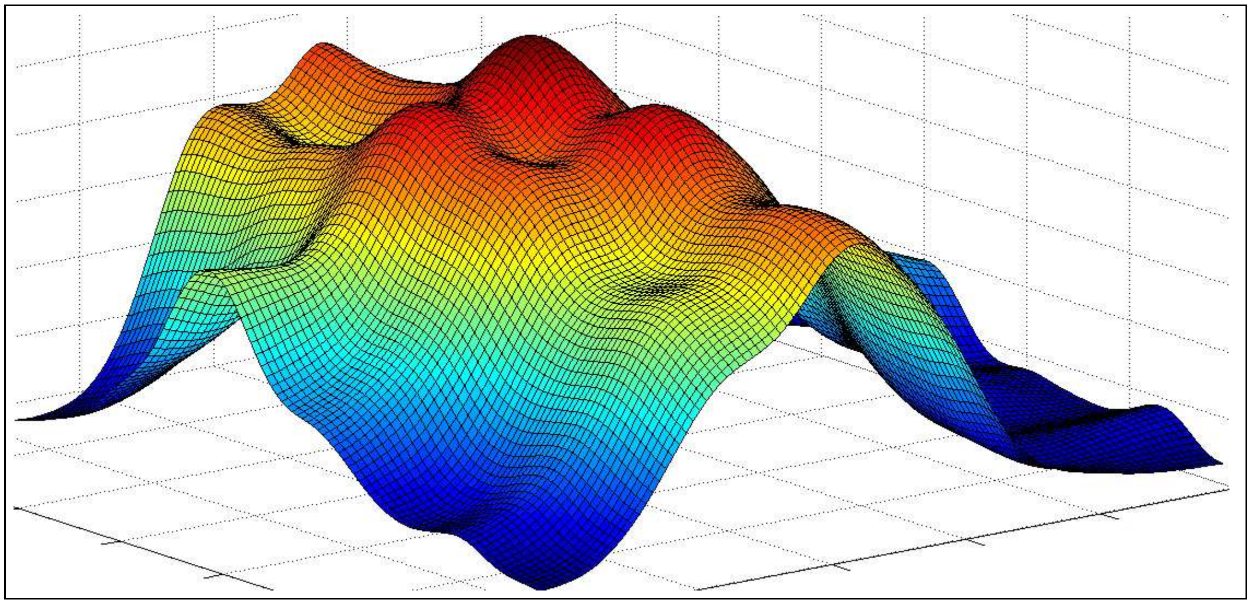


Figure 2-2: A cross section of the pose likelihood map showing the likelihood of a scan at different translations, with a fixed rotation. Notice the multiple local maxima.

MAVs is the use of the hill-climbing strategy. As can be seen in figure 2-2, which shows a 2D cross section of the 3D pose likelihood map, there are several local maxima. Unless we were lucky enough to start near the global optima, hill climbing is unlikely to find it. Vasco mitigates this problem by using the estimate from wheel odometry to initialize the hill-climbing process, however MAVs do not have that luxury.

2.2.3 Robust High-Speed Probabilistic Scan-Matching

After surveying the algorithms available for performing laser scan-matching we decided to base our algorithm on the one developed by Olson, which focused on robustness, while still managing to be computationally efficient through careful implementation. Like Vasco, Olson's method performs probabilistic scan matching using a map, however it differs in two significant ways:

1. The points in a laser scan are connected into a set of piece-wise linear contours, creating continuous surfaces in the map instead of individual points.

2. The algorithm performs a more robust exhaustive search over a set of candidate align-

ments instead of using hill climbing.

Our algorithm uses this same approach, however we made several changes to adapt it such that it could handle the $3D$ environment and run in real-time with the high resolution required by the MAV. In addition, we added a "polishing" step after the exhastive search where we use the gradient ascent method shown in algorithm 2 to refine the pose estimate. Since we start the gradient ascent from the optimum found by exhaustive search, the optimization is very fast, and we are more likely to find the global optima than if we did not perform the exhaustive search first.

Local Map Generation

To find the best alignment for an incoming laser scan, one needs a method for scoring candidate poses based on how well they align to past scans. As mentioned above, laser scanners provide individual point measurements. Successive scans will generally not measure the same points in the environment since when the laser scanner moves the measured points are shifted accordingly. Since the points do not necessarily measure the same points, attempting to correspond points explicitly can produce poor results due to incorrect matching. However, many indoor environments are made up of planar surfaces with a $2D$ cross section that is a set of piecewise linear line segments. While individual laser measurements do not usually measure the same point in the environment, they will usually measure points on the same surface. We therefore model the the environment as a set of polyline contours. Contours are extracted from the laser readings by an algorithm that iteratively connects the endpoints of candidate contours until no more endpoints satisfy the joining constraints as shown in algorithm 3.

The algorithm prioritizes joining nearby contours, which allows it to handle partially transparent surfaces such as the railings in the environment depicted by Figure 2-3(a). If we instead tried to simply connect adjacent range readings in the laser scan, there would be many additional line segments connecting points on either side of the corner. Candidate contour merges are scored and stored in a Min-heap data-structure, which allows the best candidate to be extracted efficiently. As a result, the overall contour extraction algorithm takes around 0.5ms to process a 350 point scan on modern hardware.

Algorithm 3 Extract Contours

```
Require: $S$    (set of points)
Require: $priority\_queue$ for candidate $joins$ $\{parent,child,score\}$ sorted by $score$
  $priority\_queue \leftarrow \emptyset$
  for $\mathbf{x} \in S$ do
    add $join$ of $\mathbf{x}$ with nearest free point to $priority\_queue$
  end for
  while $priority\_queue \neq \emptyset$ do
    remove best $join$ from $priority\_queue$
    if $join.parent$ already has a child then
      discard $join$
    else if connecting $join.child$ to $join.parent$ incurs too much cost then
      discard $join$
    else if $join.child$ already has a parent then
      add $join$ of $\mathbf{x}$ with nearest free point to $priority\_queue$
    else
      merge the contours of $join.child$ and $join.parent$
    end if
  end while
  return Final set of contours
```

Once we have the set of contours extracted from the previous scan, we can evaluate the likelihood of an alignment of the current scan. We assume that all point measurements in a scan are independent, and we compute the likelihood of alignment of a scan as the product of likelihoods for each individual point in the scan. As mentioned above, laser range-finders provide noisy measurements of the range and bearing of obstacles in the environment. While each of these degrees of freedom has an independent noise term, we assume a radially symmetric sensor model for simplicity. Our noise model approximates the probability of a single lidar point (x, y) as proportional to the distance d to the nearest contour C, such that

$$P(x, y|C) \propto e^{(-d/\sigma)}, \tag{2.4}$$

where σ is a variance parameter that accounts for the sensors noise characteristics. As was done for Vasco, we compute a grid-map representation where each cell represents the approximate log-likelihood of a laser reading being returned from a given location.

For most ground robotics applications, a map resolution of $10cm$ or more is often sufficient. However, to accurately estimate the *velocity* of the vehicle, where small rounding

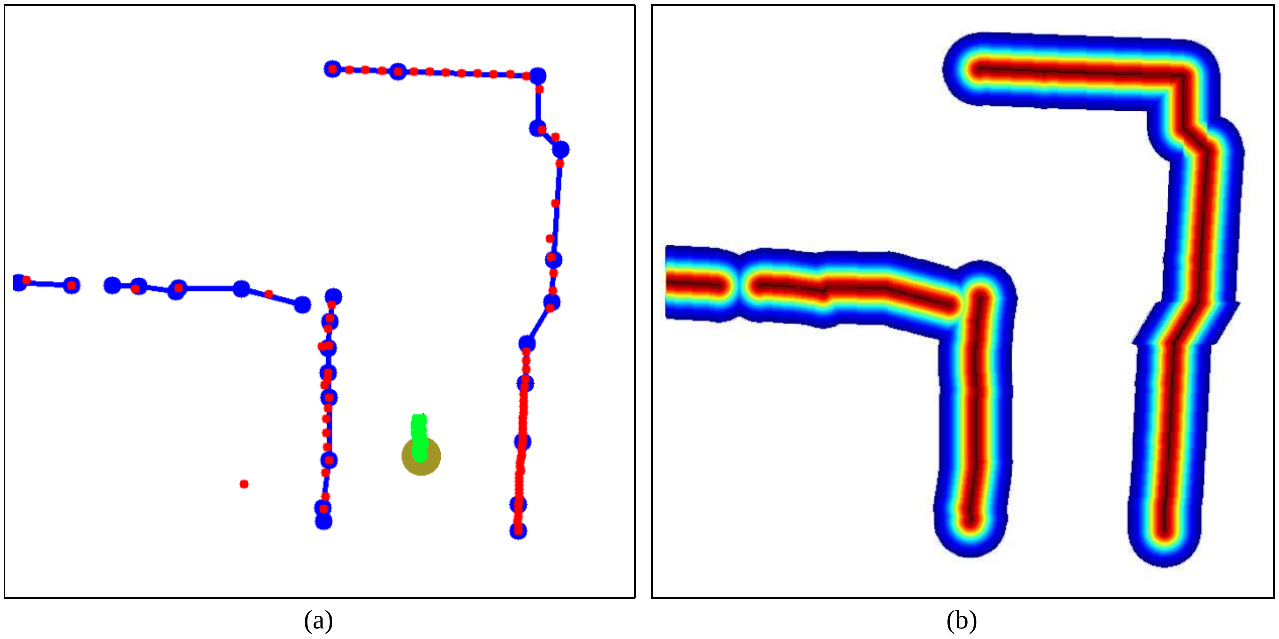


Figure 2-3: (a) Contours (blue lines) extracted from the raw laser measurements alongside the raw laser readings (red dots). Notice how the contour extraction algorithm handles the partially transparent railing on the left. (b) The resulting likelihood map generated from the contours. Brighter colors (red) indicates higher likelihood.

errors in the position are magnified significantly, we require a high resolution map with a cell size less than $1cm$. For example, if we use a map resolution of $10cm$ and the laser scans arrive at $40Hz$, an alignment that is rounded off by half a cell would result in an error on the order of $2m/s$. Since the vehicle is usually moving at less than $1m/s$ this error would be very significant. This effect is seen in the experimental results shown in Figure 2-5.

While generating these high resolution maps is computationally intensive, one can leverage their sparsity to make generating them tractable. If one examines a likelihood map such as the one shown in figure 2-3(b), one quickly realizes that with any reasonable value of σ, the vast majority of cells will be zero. So, while conventional methods compute a value for every cell in the map, and therefore require at least $O(n^2)$ operations, where n is the number of cells along an edge, we developed a likelihood map generation algorithm that exploits the sparsity in the grid map, resulting in a computational complexity of $O(m)$ where $m \ll n^2$ is the number of occupied cells.

In Olson's work, he computed the likelihood map by restricting the likelihood calculation to a local window around each line segment in the contour, and computing the distance from each pixel in the window to the line segment. While an improvement over computing the distance to the nearest line segment for the entire map, this method still does not run in real-time at the required resolution. Computing the distance to the line segment for each pixel is computationally intensive, and to make matters worse, the windows around each line segment have significant overlap, which means that each pixel ends up being modified many times.

In order to create the high resolution likelihood maps in real-time, we developed a drawing primitive that explicitly "draws" the non-zero likelihoods around each line segment. This primitive does not require us to compute the distance to the line segment for each pixel and has much less overlap in the pixels around the line segment endpoints that are touched such that in general each pixel is only written once. We accomplish this by sliding a Gaussian shaped kernel along the pixels of the line segment (as output by the Bresenham line drawing algorithm [16]), applying a $\max$ operator between the current map value and the kernel's value. Naively using a square kernel, with values set based on equation 2.4 would result in cells being written many times as the kernel slides along the line; however, one can avoid this problem by using a 1 pixel wide horizontal or vertical cross section of the kernel depending on the slope of the line. For lines that are not perfectly horizontal or vertical, this kernel must be stretched by $1/cos(s)$, where s is the slope of the line. As a final optimization, the kernel $\max$ operation can be performed using optimized matrix libraries. With the new drawing primitive, creating the likelihood map simply reduces to drawing all the line segments in the extracted contours, which takes around 20ms even for extremely large $7.5mm$ resolution likelihood maps.

We create the map from a set of k previous scans so that the relative position and velocity estimates are consistent within the local map. Comparing a new scan to an aggregate of previous scans gives much more accurate position estimates than comparing each scan only to the scan from the previous time step, as it reduces the integration of small errors over time. For example if the vehicle is stationary, the very first scan received should be the only scan in the map so that the position estimate will remain drift free. On the other hand, if we only compare each pair of incoming scans, any small errors on previous position estimates will be retained and integrated into the current position estimate, resulting in drift. In addition, since the likelihood map must be recreated every time a scan is added or removed from the map, comparing each pair of incoming scans is computationally inefficient.

In both Vasco, and Olson's work a new scan is added to the map only after the vehicle has moved some minimum distance. This heuristic is meant to ensure that scans are added to the map with enough overlap to match them effectively, while not adding scans unnecessarily frequently. The heuristic works well for ground vehicles where in general the scans only change due to motion of the vehicle. However, on a MAV the heuristic is problematic due to the drastic changes to the scan that can occur as the vehicle changes height. When the vehicle changes height, the environment may change, such that large portions of the map are quite different, despite the fact that the vehicle has not moved very far in any dimension. In this situation, we want to add the new scan to the map so that we can match against this region as well. However, if the environment has vertical walls such that the cross-section does not change as the vehicle changes height, then we do not want to add the new scan. As a result, we use a new policy whereby scans are added when they have insufficient overlap with the current map. To do this, we compute the fraction of points in an incoming scan that are above a given likelihood in the current map. When this fraction drops below a threshold, the new scan is added to the map. The threshold is set high enough that incoming scans are still able to be matched accurately, but low enough that scans are not added too often. The new heuristic reduced the amount of drift incurred by in the scan-matching process compared to the distance based one.

In addition to mitigating drift, constructing the map from multiple recent scans handles 3D effects nicely. As can be seen in figure 2-4(b), areas that vary considerably with height, such as the sides of the room, get filled in such that the entire area has high likelihood. The likelihood of a laser point being measured in those areas become almost uniform, while the likelihoods in areas that remain constant, such as the corners of the room in figure 2-4(b), are strongly peaked. Since the entire area has high likelihood its influence on the matching process will be reduced.

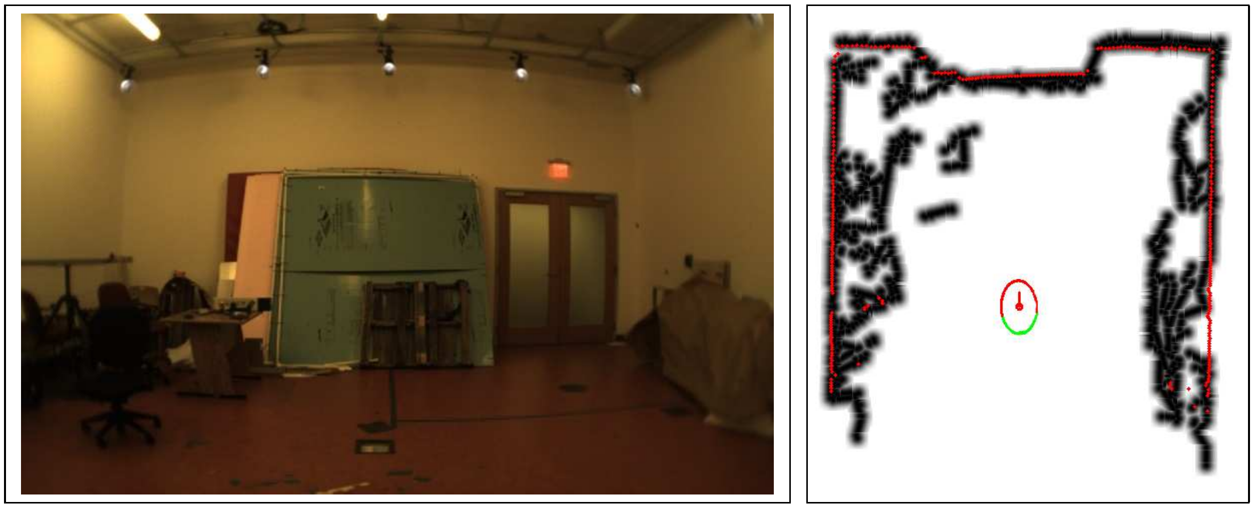

(a) (b)

Figure 2-4: (a) A cluttered lab space (b) The resulting likelihood map generated by the scan-matcher after changing heights, with the current scan overlaid. The sides of the room are very cluttered, resulting in an almost uniform distribution in some areas, while the corner remains sharply peaked and provides good alignment.

Scan-to-Map Alignment

Once we have computed the likelihood map, the second task is to find the best rigid body transform Δ^* for each incoming scan with respect to the current likelihood map. Many scan-matching algorithms such as Vasco use gradient descent techniques to optimize the values of the transform. However, as we mentioned in section 2.2.2, the three dimensional pose likelihood space is often very complicated, even for fairly simple environments. As a result, we chose to follow Olson, and use a very robust, if potentially computationally inefficient, exhaustive search over a three-dimensional volume of possible poses. The number of candidate poses in the search volume is determined by the size of the search volume (how far we search) and the translational and angular step size (how finely we search). Unfortunately, the chosen step sizes limit the resolution with which we can match the scans, so we modify Olson's approach to perform gradient ascent from the global optima chosen by the exhaustive search.

While this exhaustive search might initially seem hopelessly inefficient, if implemented carefully, it can be done very quickly. A naive implementation of this search might perform the search using three nested for-loops to iterate through all poses in the search volume. A fourth for-loop would then iterate over all points in the scan, projecting them, and looking up their likelihood in the map. Much of the computational cost in this search is taken up by performing the projection of the laser scan in the innermost loop. However, for a given candidate angle θ, the projected points in a laser scan are related by a simple translation. As a result, if we iterate over θ in the outermost loop, and perform the rotation component of the projection of each point there, the inner loops only have to perform a simple addition to complete the projection. Furthermore, if we set the resolution of the likelihood map to be equal to the translational step size, then the set of likelihoods for all translations of a test point are contained in the cells that surround the test point in the map. Iterating over the translational search window is therefore much faster since the innermost loop which iterates over the points only performs a table lookup and does not have to project the test points. As a final optimization, the entire translational search window can be accumulated into the $3D$ pose likelihood map in one step using the optimized image addition functions available in the Intel Performance Primitives [23], which provide a factor of 2 speed up.

The optimized exhaustive search implementation is considerably faster than a naive implementation, however, we still must ensure that the area over which we search is not too large. Since we do not have wheel odometry with which to initialize the scan-matching, we assume that the vehicle moves at a constant velocity between scans. With this starting point, the range of poses that must be searched over can then be selected based on the maximum expected acceleration of the vehicle, which means that at high scan rates, the search volume is manageable.

In Olson's method, the resolution with which we build the map limits the accuracy with which we can estimate the pose of the vehicle since we match the translational step size to the map resolution. However, step sizes smaller than the map resolution can change the scan likelihood due to points near the boundary between map cells being moved across the boundary. To improve the accuracy of our relative position estimates beyond the resolution used to create the map, we added a polishing step to the scan-to-map alignment where we apply the gradient ascent method described in algorithm 2 to optimize the final pose estimate. Using gradient ascent on top of the exhaustive search retains the robustness of Olson's method, while improving the accuracy considerably as shown in Figure 2-5. Since

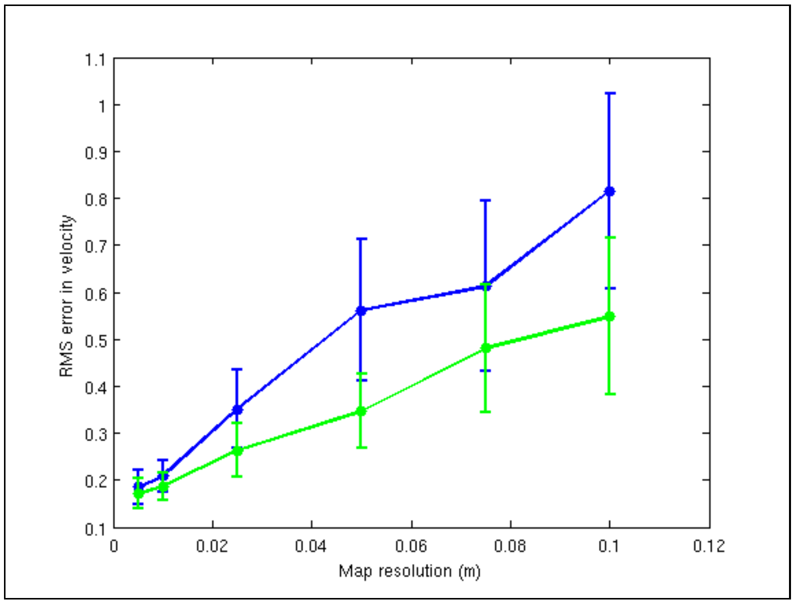


Figure 2-5: Comparison of the RMS error in velocity as a function of the map resolution, and whether gradient ascent polishing was used. The blue line shows the error for each map resolution without performing the gradient ascent polishing step, while the green line shows the same experiment with gradient ascent turned on.

we initialize the gradient ascent from the global optima found by the exhaustive search, the algorithm converges very quickly (usually $\sim 1ms$), and is likely to find the true global optima.

In our implementation, we use step sizes (and grid spacing) of 7.5mm in x, y, and $.15^{\circ}$ in θ. At this resolution, it takes approximately $5ms$ to search over the approximately $15,000$ candidate poses in the search grid to find the best pose for an incoming scan. This means that we are able to process them at the full $40Hz$ data rate. When a scan needs to be added to the likelihood map, this is done as a background computational process, allowing pose estimation to continue without impeding the real-time processing path.

Covariance Estimation

In addition to being very robust, computing the best alignment by exhaustive search has the advantage of making it easy to obtain a good estimate of the covariance in the relative-position estimates by examining the shape of the pose likelihood map around the global optimum. This estimate of the covariance is important when we integrate the relative position estimates with other sensors in the data fusion module described in section 3.2. While the entire pose likelihood map has many local maxima as shown in Figure 2-2, in the immediate vicinity of the global optima the pose likelihood is usually a fairly smooth bell shape. If the environment surrounding the vehicle has obstacles in all directions, such as in a corner, the alignment of scans will be highly constrained, resulting in a very peaked pose likelihood map. On the other hand, if the environment does not constrain the alignment, the pose likelihood map will be nearly flat at the top. An example of two such environments is shown in Figure 2.2.3.

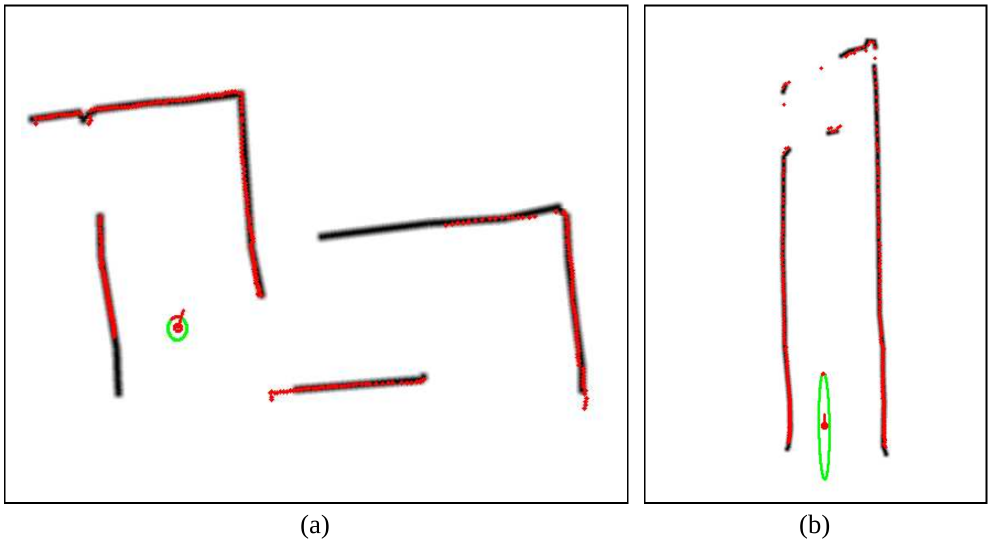


Figure 2-6: Examples of the covariance estimate output by the scan-matcher in different environments. (a) shows an environment with obstacles facing all directions which constrains the alignment of subsequent scans. (b) shows a hallway environment with very little information along the hallway.

While one could compute the covariance of the goal distribution across all degrees of freedom, for implementation simplicity we compute the covariance in rotation separately from translation, making the assumption that rotation is independent of translation. For translation we look at the $2D$ slice of the pose likelihood map at the optimal rotation. We then threshold this $2D$ map at the 95^{th} percentile, and fit an ellipse to the resulting binary image. The area and orientation of this ellipse is used as our estimate of the measurement covariance. For rotation, we find the score of the best translation for each rotation, and look at the width of the resulting bell shaped $1D$ curve. In the future, we intend to fit a multi-variate Gaussian directly to the pose likelihood map, following the example of [69].

Contributions

While the scan-matching algorithm described above is based on the original implementation by Olson, it required several modifications to enable it to be used on the MAV. Specifically, these contributions are:

1. The drawing primitive described in section 2.2.3 allows us to generate the high resolution likelihood map in real-time.
2. Our notion of a "local map" differs from [70]: scans are added based on insufficient overlap rather than distance traveled.
3. Our use of image addition primitives to accumulate the pose likelihood map.
4. Adapting the search window based on the maximum expected acceleration of the vehicle.
5. Using gradient ascent to refine the relative motion estimate is new.
6. The method for obtaining a covariance estimate is new.

2.2.4 Laser Based Height Estimation

The laser scan-matching algorithms described above only estimate the relative motion of the vehicle in x, y, and θ. In order to control the MAV we must also be able to accurately estimate the relative position in z. While the laser range-finder normally only emits beams in a $2D$ plane around the vehicle, we are able to redirect a portion of the beams downward using a right-angle mirror as shown in Figure 2-7.

Since there are so few range measurements redirected toward the floor, it is impossible to recognize distinctive features that would allow us to match measurements together to disambiguate the motion of the vehicle from changes in the height of the floor. We therefore cannot use techniques similar to the the scan-matching algorithms described above to simultaneously estimate the height of the vehicle and the floor. If we assume that the

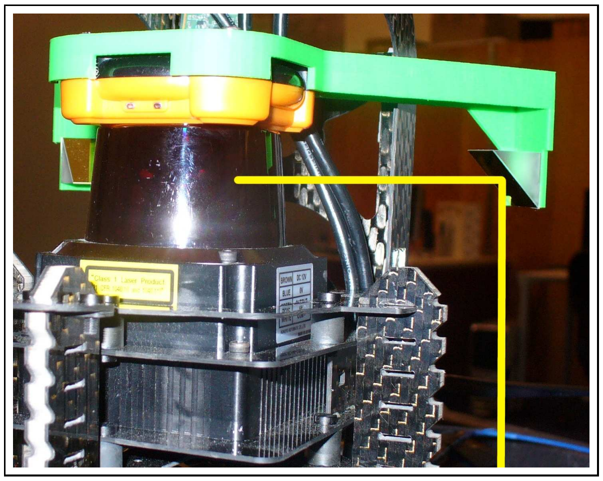
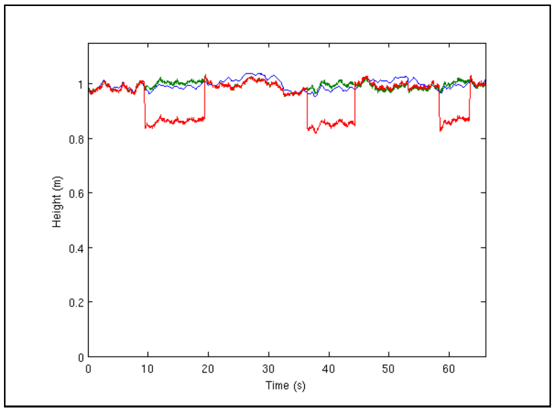
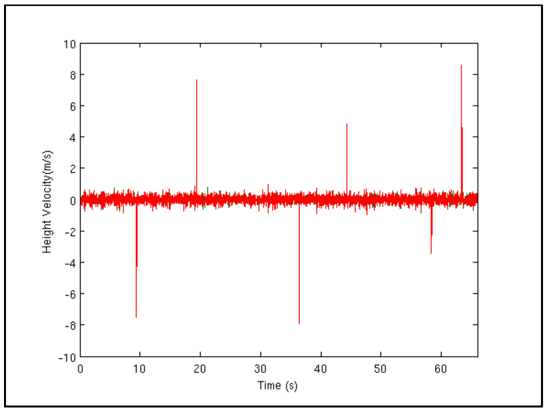


Figure 2-7: Image of the right-angle mirror used to deflect a portion of the laser beams downward to estimate height. The yellow line shows an example beam for visualization.

vehicle is flying over a flat floor, we can use the range measured by the laser scanner, r_t, directly as the estimate of the height of the vehicle at time t. However, if the vehicle flies over an object such as a table, these height estimates will be incorrect. To make matters worse, flying over the table will appear as a large step discontinuity in the height estimate, as shown in the red line in Figure 2-8(a), which can result in aggressive corrections from the position controller. However, if we look at the vehicle velocity that these height esti-



(a)



(b)

Figure 2-8: Plots showing a comparison of the height (a) and velocity (b) estimates from our algorithm (green) alongside the raw measurements (red), and the ground truth (blue). The large discontinuities in the red lines are places where the vehicle flew over the edge of an object. The object was roughly $15cm$ tall.

mates would imply, we see that there are very large outliers that occur when the vehicle flies over an object. As a result, we can use the maximum expected acceleration of the vehicle to discard changes in height that are too large, assuming that the height of the floor changed instead of the vehicle. Unfortunately, in reality, both the vehicle and the floor height change simultaneously, so simply discarding the relative height estimate will result in drift over time.

To mitigate this drift, one would ideally adopt a SLAM framework that is able to recognize when the vehicle returns to the same place and closes the loop. However, while this could be done using a camera, or another $3D$ sensor, with only the $2D$ laser scanner there is not enough information about the shape of the floor to identify loop closures. We therefore cannot rely on being able to accurately estimate the height of the floor which would then allow us to estimate our height relative to this floor estimate. Instead, we leverage the assumption that the vehicle would predominantly be flying over open floor, but sometimes fly over furniture or other objects. In this situation, over long time periods the raw height measurements from the downward pointed laser beams are usually correct, but over short time periods they may be perturbed by flying over obstacles. We therefore chose to mitigate the drift in our height estimate by slowly pushing them back towards the raw measurements r_t. While we are still above an object, we do not want to push the height estimate toward r_t since this measurement will not correspond to the actual height of the vehicle. Instead we would like to wait until the vehicle has moved away from the object and is above the floor. Unfortunately, we don't have a way to identify whether the vehicle is above the floor or a new obstacle. As a result, we make this decision based on the x-y distance the vehicle has moved since it noticed that the height of the surface underneath it changed. We choose this distance threshold by estimating the average size of objects that the vehicle will encounter. Rather than make a hard decision, we make a soft decision about when corrections should be applied and scale the magnitude of the correction by a logistic function that is centered over the average obstacle size.

More formally, our approach estimates the height of the vehicle, by integrating the difference between successive height range measurements r_t and r_{t-1} filtered such that they obey the maximum acceleration constraint. This estimate of the height will incur errors when the vehicle goes over obstacles. As a result, we apply a correction term which pushes the estimate $\hat{h}_t$ back towards the raw height, r_t, measured by the laser. The magnitude of this correction is a logistic function of the planar distance d to the last discarded height measurement.

$$\hat{h}_t = h' + \operatorname{sgn}(r_t - h')\frac{\alpha}{(1 + e^{(\sigma_d \bar{d} - d)})} \tag{2.5}$$

where h' is the height estimate after adding the filtered estimate, α is a small scaling constant on the correction term, σ_d is a parameter that controls the width of the logistic function, and $\bar{d}$ is the center of the logistic function. We chose $\bar{d}$ to be large enough that we expect the vehicle to no longer be above an object when the larger corrections are being applied. The current height estimate can be arbitrarily far away from the current measured height, which is why we use only the direction of the error in the correction, and ignore the magnitude. The correction is scaled such that the maximum correction is small enough to induce smooth motions of the vehicle. The complete height estimation process is shown in algorithm 4.

Algorithm 4 Laser Height Estimation

```
Require: $\hat{h}_{t-1}$    (previous height estimate)
Require: $r_t, r_{t-1}$    (current and previous height measurements)
Require: $v_{t-1}$    (previous velocity estimate)
Require: $d$    (distance to previous discarded measurement)
  if $|(r_t - r_{t-1})/dt - v_{t-1}| > MAX\_ACCELERATION$ then
    $h' = \hat{h}_{t-1}$
    $d = 0$
  else
    $h' = \hat{h}_{t-1} + (r_t - r_{t-1})$
  end if
  $\hat{h}_t = h' + \operatorname{sgn}(r_t - h')\alpha/(1 + exp(\sigma_d \bar{d} - d))$
  return $\hat{h}_t, d$
```

2.3 Visual Odometry

While the laser-based relative position estimation solution is efficient and works surprisingly well in $3D$ environments despite the $2D$ nature of the sensor, using a camera sensor to compute relative position estimates is attractive for increased robustness in the face of significant $3D$ structure in the environment. Despite this appeal, camera data can be much more difficult to work with due to the lack of range information, and the large amount of raw data. We have developed a stereo-vision based visual odometry solution which manages to overcome these difficulties to provide relative position estimates that are sufficient to stabilize the MAV. This section provides an overview of the developed solution. For a more complete description and analysis of the algorithm, I refer the reader to Markus Achtelik's thesis [6].

In general, a single camera is sufficient for estimating the relative motion of the vehicle by corresponding features from consecutive image frames. If enough feature correspondences (at least 7-8) are available, the fundamental matrix describing the motion of the camera can be computed. Decomposing this matrix yields the relative rotation and translation motion of the camera. However, this estimate cannot resolve the scale of the motion, just the relative degree of different motions [61]. While the rotation can be uniquely computed with respect to this scalar factor, we can only compute the translational direction of the motion, and not its magnitude.

To resolve this scale ambiguity, either scene knowledge is necessary, or two successive views must have distinct vantage points and a sufficiently large baseline between them, since motion in the direction of the camera's optical axis cannot be computed accurately. Given that scene knowledge for unknown environments is typically unavailable, and MAVs often move slowly and forward with the camera facing front, autonomous navigation for MAVs with a signle camera has proven to be very challenging (refer to section 1.3). This motivated our choice of a stereo camera to reconstruct the 3D-position of environmental features accurately. The stereo-rig not only enforces a baseline distance between the camera views, but also allows us to reconstruct the feature positions in a single time step, rather than using consecutive frames from a monocular camera. In this section, *left* and *right* denote the images taken from the left and right stereo cameras respectively, as seen from the MAV's frame of reference.

Our approach for stereo visual odometry is outlined in figure 2-9. Image features are first detected in the *left* frame from the previous time-step (1). These features are then found in the previous *right* frame (2), enabling us to reconstruct their positions in 3D-space using

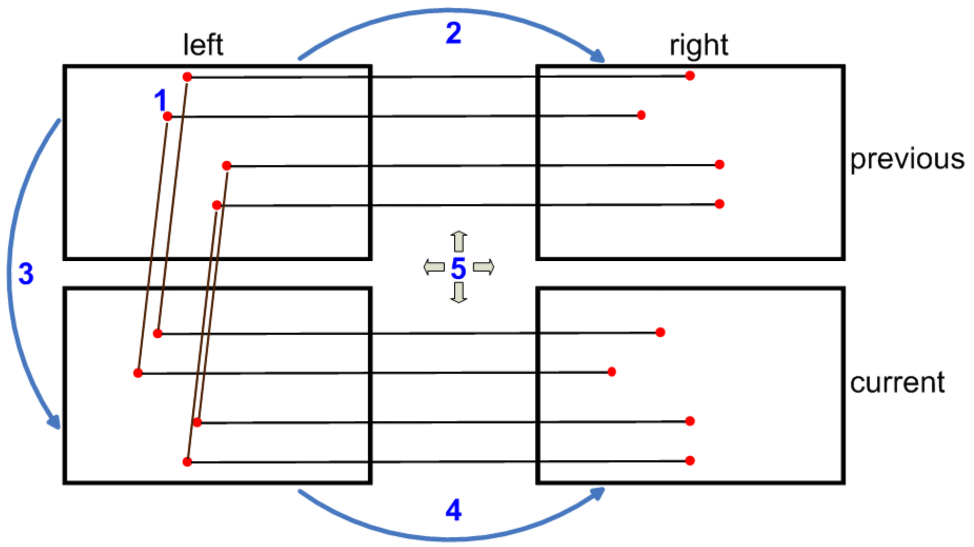


Figure 2-9: Steps performed on each frame in the stereo visual odometry algorithm: 1) Perform feature detection. 2) Find correspondences between *left* and *right* frame for depth reconstruction. 3) Find correspondences between the previous and current frames. 4) Repeat step 2 on current frames. 5) Frame to frame motion estimation.

triangulation. Successfully reconstructed features are then tracked from the previous *left* to the current *left* frame (3), and a similar reconstruction step is performed for the current frames (4). This process results in two “clouds” of features that relate the previous and current views. With these correspondences, the quadrotor’s relative motion in all 6 *dof* can be computed in a least-squares sense in closed form (5).

2.3.1 Feature detection

Image features are locations in the image which are distinctly recognizable such that finding the correspondence between the same features in different images is possible. In general image features can be any visually distinct pattern in the image, however for our purposes we seek to find “corner-features” which are characterized by strong intensity gradients in two directions. Feature detection is a basic image processing primitive underlying many computer vision algorithms, and there are many options available which have different performance and computational characteristics. Although SIFT [60] and SURF [13] features are popular choices for visual feature detectors due to the ease with which they can be corresponded, computing them fast enough for our purposes on modern hardware remains computationally infeasible given the control requirements of our quadrotors discussed in section 1.1. Instead, we adopted the FAST (Features from Accelerated Segment Test) [76] feature detector, which was introduced by Rosten et al. The FAST feature detector provides a good balance between fast speed, and feature quality. In order to avoid detecting many distinct features that are in close geometric proximity to each other, we down-sample the image before running the feature-detector, and project the feature locations back to the full size image after detection. Unfortunately, many spurious features are still detected by FAST, leading to inaccurate feature-tracking results. We therefore filter the FAST features using their Harris corner-response [40]. The use of the FAST detector beforehand allows us to compute the more computationally intensive image-gradients and corner-responses used by the Harris detector only for the areas detected by the FAST detector – a small subset of the entire image.

Unfortunately, the combination of FAST and Harris metrics might still result in features that are located in close proximity with other features even for low-resolution images. Spatial redundancy of features incurs unnecessary computational cost during the feature matching from image to image and makes feature-tracking error-prone. We therefore prune our feature set by computing the distance between all feature pairs, eliminating one of the features if the distance is less than a specified threshold. While this process theoretically has a complexity of $O(n^2)$, the computation in practice is much faster because the features are already presorted by the FAST feature detector. The feature pruning take 3-4ms for about 500 feature candidates and leaves around 150 valid features. After pruning out all the undesired features, the remaining feature locations are refined to sub-pixel accuracy using the OpenCV function cvFindCornerSubPix which requires an additional 6ms.

2.3.2 Feature Correspondence

To correspond the features between the *left* and *right* frames, as well as from the previous to the current frames, we use the pyramidal implementation of the KLT optical flow tracker available in OpenCV [15]. This implementation allows us to track features robustly over large baselines and is robust to the presence of motion blur. For correspondences between the *left* and *right* frames, error-checking is done at this stage by evaluating the epipolar constraint

$$x_{right}^T F x_{left} = 0 \pm \epsilon, \tag{2.6}$$

where x denotes the feature location in the respective frame, F is the fundamental matrix pre-computed from the extrinsic calibration of the stereo rig, and ϵ is a pre-defined amount of acceptable noise.

2.3.3 Frame to frame motion estimation

Once we have the two sets of image features in both the previous and current pair of images with known correspondences, the features are projected to a 3D-space by triangulation between the left and right cameras. Given two estimates of 3D-feature locations taken at different times, we can estimate the relative motion from the previous to the current time using the closed form method proposed by Umeyama [97]. This method computes rotation and translation separately, finding an optimal solution in a least squares sense. Unfortunately, least square methods are sensitive to outliers, and we therefore use Umeyama's method to generate a hypothesis for the robust MSAC estimator [94], a refinement of the popular RANSAC method. After finding a hypothesis with the maximum inlier set, the solution is recomputed using all inliers.

2.3.4 Nonlinear motion optimization

As mentioned in the laser scan-matching section, small measurement errors will accumulate over time, resulting in drifting position estimates over time. However, because many of the visual features remain visible across more than two consecutive frames, we can estimate the vehicle motion across several frames to obtain more accurate estimates. This can be done using bundle adjustment [96] (shown in figure 2-10), where the basic idea is to minimize the following cost function:

$$c(\mathbf{X}_i, R_j, t_j) = \sum_{i=0}^{m} \sum_{j=0}^{n} E(\mathbf{x}_{ij}, P_j \mathbf{X}_i)^2 \quad \text{with} \quad P_j = \begin{bmatrix} K_j R_j & K_j t_j \end{bmatrix} \tag{2.7}$$

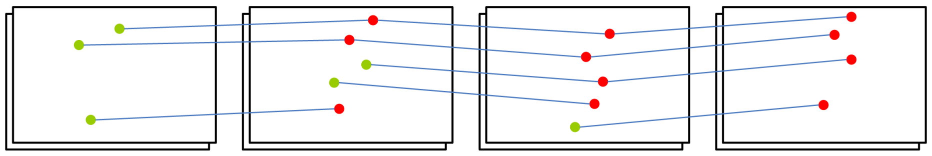


Figure 2-10: Bundle adjustment incorporates feature correspondences over a window of consecutive frames.

where $E(\mathbf{x}_{ij}, P_j\mathbf{X}_i)$ is the re-projection error due to the projection of the 3D-feature, $\mathbf{X}_i$, onto the camera's image plane using the j-th view, so as to obtain the 2D-point, $\mathbf{x}_{ij}$. Here, m and n are the number of 3D-features and views respectively, while K_j is the intrinsic camera matrix, which is assumed to be constant.

We are therefore seeking to find the optimal arrangement of 3D-features and camera-motion parameters so as to minimize the sum of the squared re-projection errors. This problem can be solved using an iterative nonlinear least squares methods, such as the technique proposed by Levenberg-Marquardt. Although the cost function in equation 2.7 appears simple, the problem has a huge parameter space. We have a total of $3m + 6n$ parameters to optimize – three parameters for each 3D-feature and six parameters for each view. In addition, in each Levenberg-Marquardt step, at least m re-projections have to be computed per view. Computation in real-time therefore quickly becomes infeasible with such a large parameter space. Fortunately, the problem has a sparse structure, since each 3D-feature can be considered independent, and the projection of X_i into x_{ij} depends only on the j-th view. This sparse-bundle-adjustment problem can be solved using the generic package by Lourakis and Argyros [59], which we used for our application.

The standard bundle adjustment approach is susceptible to outliers in the data, and because it is an iterative technique, good initial estimates are needed for the algorithm to converge quickly. We avoid both problems by using the robust frame-to-frame MSAC motion estimates, as described in section 2.3.3, as well as their inlier sets of 3D features.

Running bundle adjustment over all frames would quickly lead to computational intractability. Instead, we pursue a sliding-window approach, bundle-adjusting only a window of the latest n frames. By using the adjustment obtained from the old window as an initial estimate of the next bundle adjustment, we ensure that the problem is sufficiently-constrained while reducing the uncertainties due to noise. The presence of good initial estimates also reduces the number of optimization steps necessary. Performing bundle adjustment for 150 features using a window size of $n = 5$ takes approximately 30-50ms.

The feature correspondences necessary for bundle adjustment are found by chaining the matches from frame-to-frame. The number of features diminishes over successive frames, so new features are added at every time step. When the new features are located close to old features, the old ones are preferred.

2.4 Comparison

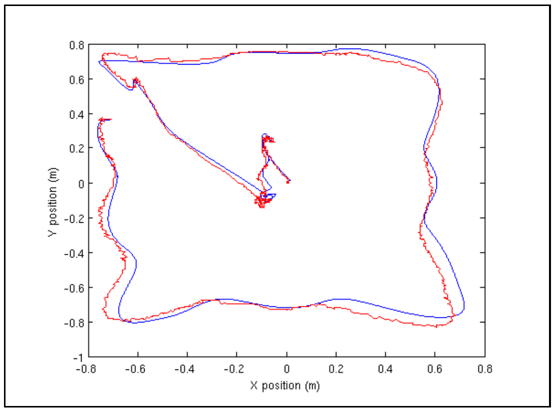
While both the laser and vision based relative position estimation algorithms described above provide estimates that enable indoor flight when combined with the rest of the system described in chapter 3, they are very different solutions to the same problem, with different performance characteristics. Overall, we found that the state estimates obtained from the laser data tended to have less drift as can be seen in figure 2-11. In addition, the laser data requires less time to transmit over the wireless and the scan-matching algorithm is able to process incoming scans at a faster rate resulting less delay in its position estimates as shown in Table 2.1. The measurement delay is critical for stable flight. Finally, the

	Computation Time	Bandwidth
Laser Scan-Matching	5 ms/measurement	170 KB/s
Visual Odometry	65 ms/measurement	1300 KB/s

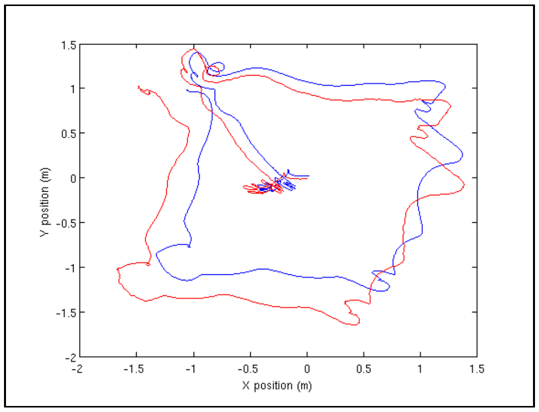
Table 2.1: Comparison of the computational and bandwidth requirements of the laser and vision based relative position estimation algorithms.

laser scan matching handles fast motion more easily. During fast motion, the cameras can experience significant motion blur, corrupting the state estimates. Much of this difference is due to inherent differences in the active vs. passive sensing modalities. With sufficient light, the exposure time for the stereo cameras could be reduced to mitigate the effect of fast motion, however that is unlikely to be possible in general environments without much larger lenses and a high power flash.

While the laser scan-matching currently seems to provide better performance, the visual odometry natively provides estimates for all 6 degrees of freedom, which is a major



(a) Laser scan-matching



(b) Visual odometry

Figure 2-11: Comparison of the relative position estimates from laser scan-matching and visual odometry (Red) against ground truth (Blue) on similar trajectories.

advantage. The laser scan-matching shows remarkable resilience to $3D$ structure in the environment, however, it is unlikely that it will work in completely unconstrained environments, such as a cave.

Both methods experience failure modes when there are insufficient environmental features to perform data association. The laser scan-matching fails in long hallways, where readings in different places along the hallway are indistinguishable. On the other hand the visual odometry will fail in areas without distinctive intensity features, such as when looking at a blank wall. Fortunately, these failure modes result from very different environmental structures, which would indicate that the best performance would be obtained by using both relative position estimation algorithms simultaneously.

Chapter 3

System Modules

This chapter describes the complete system that enables autonomous flight in indoor environments. Each module is a critical piece of the overall system, and required for safe and robust operation. The modules described in this chapter are highlighted in the blue rectangle of the system diagram, repeated in figure 3-1.

The planning and exploration module described in section 3.5 was developed in collaboration with Ruijie He.

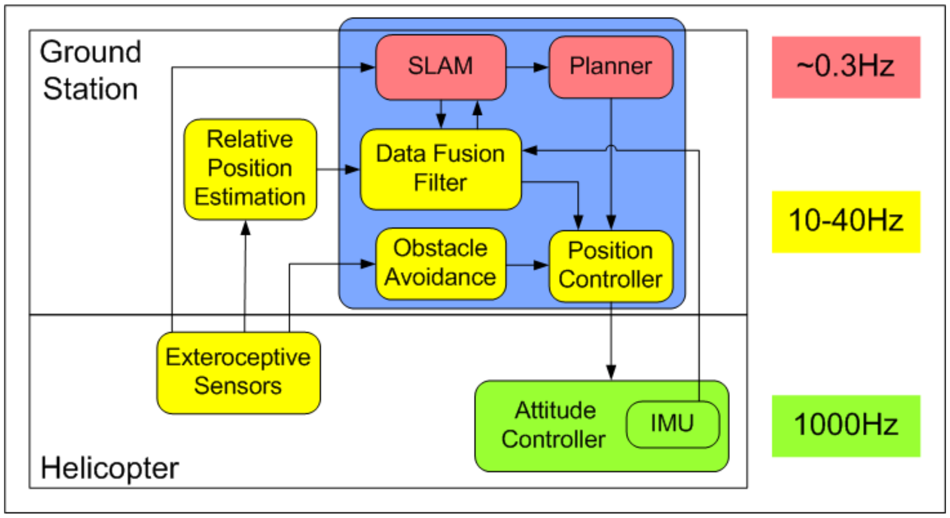


Figure 3-1: Schematic of our hierarchical sensing, control and planning system repeated for the reader's convenience. At the base level, the onboard IMU and controller (green) creates a tight feedback loop to stabilize the MAV's pitch and roll. The yellow modules make up the real-time sensing and control loop that stabilizes the MAV's pose at the local level and avoids obstacles. Finally, the red modules provide the high-level mapping and planning functionalities.

3.1 Communication

We developed the hierarchical software system shown in figure 3-1 as a set of independent modules. Each software module runs in a separate process either onboard the MAV, or at the ground station. The processes communicate using the Lightweight Communications and Marshalling (LCM) library [4, 48], originally developed for team MIT's entry into the DARPA Urban Challenge [57]. LCM uses UDP multicast to provide efficient, low-latency communication without requiring a centralized hub. The lack of a centralized hub is particularly important since it allows a separate cluster of processes to run onboard the MAV, without requiring all messages to traverse the wireless link. However, by itself, LCM does not handle transmissions over wireless links, since it was designed for use on high bandwidth dedicated networks, and therefore does not seek to correct for lost or dropped packets, expecting these events to be rare or due to congestion. Unfortunately wireless networks experience random drops due to interference or other factors relatively frequently, which makes error handling critical.

Instead of using the default UDP multicast scheme used in LCM over the wireless link between the MAV and the ground station, we create a point-to-point tunnel that encapsulates the LCM messages for transmission over the wireless link. While using a standard TCP stream for this tunnel would seem to be an obvious way to achieve reliable transmission, TCP is known to provide poor performance in wireless environments [99] due to the assumption that dropped packets are a signal of congestion rather than random errors. As a result, when TCP experiences a dropped packet, it will slow down its transmission rate in an attempt to be fair to other flows. This behavior is undesirable on a dedicated wireless link since it reduces the throughput and adds a significant amount of latency for some packets. As a result, we instead turn to using a UDP datastream with forward error correction (FEC) [67] applied for reliability. FEC adds a set of parity packets to the datastream that can be used to reconstruct lost packets. The packets add a constant amount of overhead, however as long as the aggregate bandwidth required does not overload the wireless link, this solution provides low latency delivery of the time critical sensor data. To encode the data stream we use low density parity check codes which provide high efficiency in both the amount of overhead and computation required to encode and decode the data.

3.2 Data Fusion Filter

While the relative position estimation algorithms described in the previous chapter provide accurate and timely relative position estimates, to compute the full MAV state estimate, including the velocities, we use an Extended Kalman Filter (EKF) to fuse the relative position estimates with the acceleration readings from the IMU. This setup is inspired by the standard practice for making use of GPS and an IMU on outdoor UAVs [51]. Using a data fusion filter has several advantages over directly using the relative position estimates and their derivatives to control the vehicle. While the IMU readings drift significantly and are therefore not useful over extended time periods, they are useful over short time periods and allow us to improve our estimate of the vehicle velocities.

Another advantage to using the EKF for data fusion is that it provides a clean way to interpolate the state estimates used by the feedback controller. This is particularly important since the wireless link and processing time adds a variable delay to the raw measurements, which can cause problems for controlling the vehicle. Therefore, in our EKF formulation, we perform the measurement updates asynchronously whenever they arrive, while the motion model prediction step is performed on a fixed clock. This setup is also robust against measurements that get lost completely due to wireless interference.

3.2.1 Extended Kalman Filter Background

The Kalman filter [53] is an efficient recursive filtering algorithm for estimating the probability distribution over the state of a dynamic system, called the belief state, from a stream of noisy measurements. The algorithm alternates between a *process* update, where the next state is predicted from the current state and control input, and a *measurement* update where a sensor measurement corrects the current belief. In the special case where the state transition and observation models are linear and subject to Gaussian noise, the Kalman filter provides the optimal belief estimates for the system in a least-squares sense.

However, in many real-world applications, the assumption that the observations are linear functions of the state and that the state transition function is a linear function of the state and control breaks down. For example, a simple robot making translational and rotational actions cannot be described by linear state transitions. Plain Kalman filters are therefore inapplicable to many robotics problems, such as estimating the state of our MAV.

The extended Kalman filter was developed to overcome some of these fundamental limitations of the pure Kalman filter. The EKF allows the same inference algorithm to operate with non-linear transition and observation functions by linearizing these functions around the current mean estimate. More formally, let the state s_t and observation z_t at time t be given by the following functions:

$$s_t = g(s_{t-1}, u_t, w_t), \qquad w_t \sim N(0, W_t), \tag{3.1}$$

$$\text{and} \qquad z_t = h(s_t, q_t), \qquad q_t \sim N(0, Q_t), \tag{3.2}$$

Here, u_t refers to the control action, and w_t and q_t are random, unobservable noise variables. The functions g and h represent the non-linear control and measurement models respectively.

In the EKF, the belief state is represented by a Gaussian distribution, parameterized by a mean estimate, μ_t, and a covariance matrix, Σ_t, which represents the associated uncertainty. The EKF computes the state distribution at time t in two steps: a process step that is based only on the control input u_t and the belief state in the previous time step, (μ_t, Σ_t), as well as a measurement step that incorporates measurement z_t to obtain the new belief estimate.

The process step is calculated as follows:

$$\overline{\mu}_t = g(\mu_{t-1}, u_t), \tag{3.3}$$

$$\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + V_t W_t V_t^T, \tag{3.4}$$

where G_t is the Jacobian of g with respect to x and V_t is the Jacobian of g with respect to w. For convenience, we denote $R_t \triangleq V_t W_t V_t^T$. The result is the predicted belief state, represented by the predicted mean estimate $\overline{\mu}_t$ and predicted covariance $\overline{\Sigma}_t$.

Algorithm 5 The Extended Kalman Filter algorithm

```
Require: Previous belief state $\mu_{t-1}, \Sigma_{t-1}$, action $u_t$, observation $z_t$
1: $\overline{\mu}_t = g(u_t, \mu_{t-1})$
2: $\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
3: $K_t = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1}$
4: $\mu_t = \overline{\mu}_t + K_t(z_t - h(\overline{\mu}_t))$
5: $\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t$
6: return $bel(\mu_t, \Sigma_t)$
```

Similarly, the measurement step is calculated as follows:

$$\mu_t = \overline{\mu_t} + K_t(H_t \overline{\mu}_t - z_t), \tag{3.5}$$

$$\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t, \tag{3.6}$$

$$K_t = \overline{\Sigma}_t H_t^T \left(H_t \overline{\Sigma}_t H_t^T + Q_t\right)^{-1}. \tag{3.7}$$

where H_t is the Jacobian of h with respect to s. K_t is known as the Kalman gain, which represents the mapping of the measurements z_t from measurement space to state space that will yield the Minimum Mean-Square Error (MMSE) estimate. The EKF algorithm is summarized in Algorithm 5.

3.2.2 Process Model

Our filter is a standard EKF, implemented using the open source KFilter library [3]. We use the filter to estimate the positions, velocities, and accelerations of the vehicle, along with the biases in the IMU. More specifically we represent the state as:

$$\mathrm{s} = [x^g, y^g, z^g, \theta, \phi, \psi, \dot{x}^b, \dot{y}^b, \dot{z}^b, \dot{\theta}, \dot{\phi}, \dot{\psi}, \ddot{x}^b, \ddot{y}^b, \ddot{z}^b, bias^{\ddot{x}}, bias^{\ddot{y}}, bias^{\ddot{z}}, bias^{\phi}, bias^{\psi}] \tag{3.8}$$

The position and orientation $(x^g, y^g, z^g, \theta, \phi, \psi,)$ are represented in the global coordinate frame, with the origin placed where the vehicle is initialized, and angles represented using Euler angles. The velocities and accelerations are represented in the body frame, where the origin is located at the center of body with the x-axis pointing towards the front of the vehicle, y to the left, and z up.

In section 3.3 we present a dynamics model used for control, which models the relation between the control inputs and the vehicle accelerations. While this model is sufficiently accurate for control, the errors are sufficiently large compared the measurements from the IMU that for simplicity we do not include the control inputs in the process model. As a result, the nonlinear state update equations for x_g and y_g are:

$$\begin{aligned} x^g_t &= x^g_{t-1} + \Delta t(\dot{x}^b_{t-1}\cos(\theta_{t-1}) - \dot{y}^b_{t-1}\sin(\theta_{t-1})) + \omega_x, && \omega_x \sim N(0,\sigma_x) \\ y^g_t &= y^g_{t-1} + \Delta t(\dot{x}^b_{t-1}\sin(\theta_{t-1}) + \dot{y}^b_{t-1}\cos(\theta_{t-1})) + \omega_y, && \omega_y \sim N(0,\sigma_y) \end{aligned} \tag{3.9}$$

where Δt is the filter update period, and $\omega_{x,y}$ are zero mean Gaussian noise terms. The rest of the position and the linear velocity states are updated by discrete integration:

$$\mathbf{v}_t = \mathbf{v}_{t-1} + \Delta t\mathbf{v}_{t-1} + \omega_{\mathbf{v}}, \quad \omega_{\mathbf{v}} \sim N(0,\sigma_{\mathbf{v}})$$

where $\mathbf{v} = [z^g, \theta, \phi, \psi, \dot{x}^b, \dot{y}^b, \dot{z}^b]$ is a sub-vector of the vehicle state s. Finally, the angular velocities, linear accelerations, and bias terms are updated with a zero mean Gaussian noise term :

$$\mathbf{a}_t = \mathbf{a}_{t-1} + \omega_{\mathbf{a}}, \qquad \omega_{\mathbf{a}} \sim N(0,\sigma_{\mathbf{a}}) \tag{3.10}$$

where $\mathbf{a} = [\dot{\theta}, \dot{\phi}, \dot{\psi}, \ddot{x}^b, \ddot{y}^b, \ddot{z}^b, bias^{\ddot{x}}, bias^{\ddot{y}}, bias^{\ddot{z}}, bias^{\phi}, bias^{\psi}]$.

3.2.3 Measurement Model

To incorporate the measurements into the filter, we created a modular design that allows us to use the same basic filter setup for different combinations of sensors. The IMU is treated as if it measures the accelerations and attitude plus the associated bias.

$$\mathbf{z}^t_{IMU} = \begin{bmatrix} \ddot{x}^t_b + bias^t_{\ddot{x}} \\ \ddot{y}^t_b + bias^t_{\ddot{y}} \\ \ddot{z}^t_b + bias^t_{\ddot{y}} \\ p^t + bias^t_p \\ r^t + bias^t_r \end{bmatrix} \tag{3.11}$$

Adding the relative position estimates to the filter is not quite as straightforward since we much choose whether to treat the estimates as either position or velocity measurements. The relative position estimation modules measure the distance between sensor measurements, which gives us the option to either integrate these relative measurements into position measurements, or divide by the time between measurements to get velocity measurements. In general, since the wireless link can add an unknown delay to the sensor measurements we prefer to integrate the relative position estimates given by either laser scan-matching or visual odometry alone as position measurements. However, when we want to fuse all three sensing modalities, laser, vision, and inertial, we cannot add both the laser and vision as positions since they will not necessarily be consistent with each other and may be in very different coordinate systems. This is due to the fact that the position estimates of each sensor are the result of an integration, so any errors will be propagated along indefinitely. This is particularly problematic if there are errors in θ, which results in the incoming measurements being in different coordinate frames. If this happens, we do not want the state estimate to be a weighted average of the two coordinate-system estimates, which is what the filter would compute if we naively add both as position measurements without rotating the coordinate frames into alignment. If we knew the transformation between the two coordinate-systems, we could transform the measurements before adding the measurements to the filter, however, we do not have access to this transformation. We instead choose to add the laser scan-matching estimates as position measurements, and the visual odometry estimates as velocity measurements. The laser scan-matching does not measure pitch and roll, so the laser measurements are given as

$$\mathbf{z}^t_{POS} = [x^t_g, y^t_g, z^t_g, \theta^t]^\top \tag{3.12}$$

whereas the visual odometry measures all 6 degrees of freedom

$$\mathbf{z}^t_{VEL} = [\dot{x}^t_b, \dot{y}^t_b, \dot{z}^t_b, \dot{\theta}^t, \dot{p}^t, \dot{r}^t]^\top \tag{3.13}$$

We chose to use the laser odometry for position because it usually had less drift than the visual odometry estimate.

The remaining design work for the data fusion filter was to determine reasonable variance parameters for the process and measurement models. We chose to make all the process and measurement covariance matrices diagonal to limit the number of parameters that needed to be tuned. For the measurement covariances of the relative position estimates, we assumed the variance was a scalar multiple of the estimated covariance computed by the odometry module described in section 2.2.3. Despite the simplifying assumption of diagonal covariances, there were still a very large number of parameters that must be tuned to achieve good filter performance. Doing this process by hand would have been a very time consuming and error prone process, so we learned the variance parameters using a method similar to the one described in [5].

3.2.4 Filter Tuning

To learn the variance parameters, we make use of the highly accurate state estimates available when flying inside of our Vicon [98] motion capture environment. The motion capture system consists of a calibrated array of cameras which track reflective markers placed on the vehicle. The system provides the sub-millimeter accurate position and orientation estimates of the vehicle at $120Hz$, which can be considered "ground truth" data. We collect a log containing the data fusion module's input data d as given by $\mathbf{z}_{IMU}$, $\mathbf{z}_{POS}$, and $\mathbf{z}_{VEL}$ alongside the ground truth state values s^* from motion capture. We then replay the log to compute the output of the EKF with parameter vector $\varTheta$, containing the variance parameters σ from equations 3.9, 3.10 and 3.10:

$$\tilde{s} = K(d; \varTheta) \tag{3.14}$$

To evaluate the filter performance we must choose a cost function, that penalizes deviations from the ground truth values. In the setting described in Abbeel et al's work [5], they used a high quality GPS unit as the ground truth measurements for a lower quality GPS unit. They compared several cost functions and found that the RMS error between the state estimates computed with the lower quality GPS and the ground truth provided the best results. In our setting, the ground truth position estimates will not be in the same coordinate frame as our filter's estimate due to drift in the relative position estimates, as discussed above. This will cause a small error at the beginning of the log to incur a disproportionately large cost compared to an error near the end of the log. Since there is no reason to prefer errors at one time versus another, we must treat the errors in x, y, and θ differently from the rest of the states. For these states, we penalize errors in relative motion $\Delta_t = [\Delta_t^x, \Delta_t^y, \Delta_t^\theta]$ rather than absolute position.

$$\Delta_t^x = (x_t - x_{t-1})\cos(\theta_{t-1}) - (y_t - y_{t-1})\sin(\theta_{t-1}) \tag{3.15}$$

$$\Delta_t^y = (x_t - x_{t-1})\sin(\theta_{t-1}) + (y_t - y_{t-1})\cos(\theta_{t-1}) \tag{3.16}$$

$$\Delta_t^\theta = \theta_t - \theta_{t-1} \tag{3.17}$$

Since only a subset of the states are used for control (the rest are estimated to improve the estimates of the others), for $s_{rest} = [z, p, r, \dot{x}, \dot{y}, \dot{z}]^\top$, we compare the values directly, and ignore the remaining states. Finally, to reduce over-fitting, and encourage smooth state estimates we incorporate a smoothness term in our cost function. The resulting cost function for time t is:

$$C^t(\tilde{s}_t, \tilde{s}_{t-1}, s_t^*, s_{t-1}^*) = ||\tilde{\Delta}_t - \Delta_t^*||_2 + ||\tilde{s}_{rest} - s_{rest}^*||_2 + ||\tilde{s}_t - \tilde{s}_{t-1}||_2 \tag{3.18}$$

The cost for the entire log $C(\tilde{s}, s^*)$ is the summation of the cost for each individual time step. Since each of the state variables have different units and scales, we also include weighting parameters that balance out the contribution of each state.

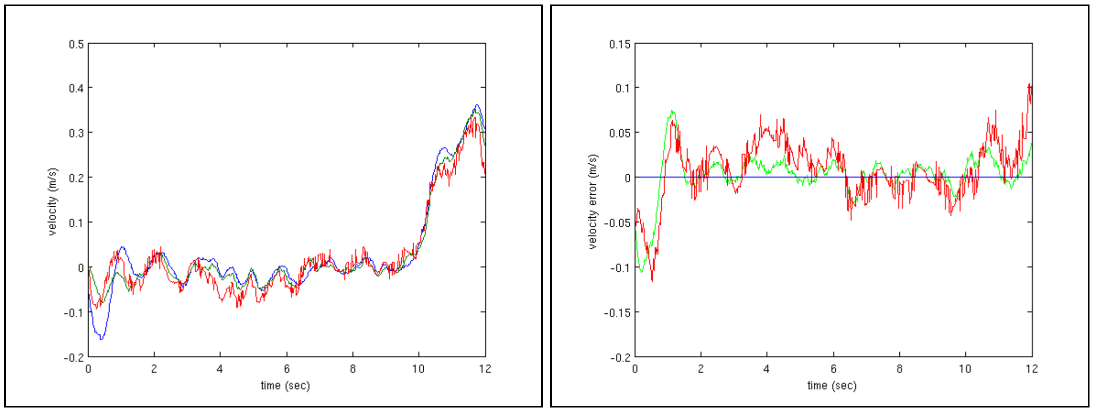
We then use stochastic gradient descent (SGD) to optimize the variance parameters with respect to this cost function. SGD is particularly attractive since it does not require us to explicitly compute the performance gradients, which would be quite challenging due to the complex interactions of the variance parameters with the estimated states, and the nonlinear cost function. The SGD algorithm follows the cost surface gradient in expectation and is guaranteed to converge to a local optima. While other optimization methods might be faster, the optimization is a one-time cost, so ease of implementation was chosen over performance. The final parameter filter tuning algorithm is presented out in algorithm 6. We initialize the variance parameters by computing the variance of the difference between the input measurement data and the ground truth estimates.

Algorithm 6 The Filter Tuning Algorithm

```
Require: $d$ (the data to be filtered)
Require: $s^*$ (ground truth state estimates)
Require: $\sigma, \eta$ (noise and learning rater parameters)
Require: $\mathit{\Theta}_b \leftarrow \Theta_0$ (initial guess for variances)
  while Average Cost Decreases do
    #perturb parameters:
    $\mathit{\Theta}_p = \mathit{\Theta}_b + \mathcal{N}(0, \sigma)$
    #compute state estimates:
    $\tilde{s}_b = K(d; \mathit{\Theta}_b)$
    $\tilde{s}_p = K(d; \mathit{\Theta}_p)$
    #compute costs:
    $c_b = C(\tilde{s}_b, s^*)$
    $c_p = C(\tilde{s}_p, s^*)$
    #update variances:
    $z = \mathit{\Theta}_p - \mathit{\Theta}_b$
    $\Theta_b = \Theta_b - \alpha(c_b - c_p)z$
  end while
```

The parameter learning algorithm generally converges after a couple hours of computation time depending on the length of the input log used. It is able to significantly reduce the error on a held out test log as shown in figure 3-2. Overall, the tuned EKF produces state estimates that are significantly more accurate than the filter performance obtained with variance parameters chosen by hand. For example, on several logs of real flight data, the average velocity error from the filter using learned parameters was half that of the filter using hand-chosen parameters. In addition, and perhaps more importantly, the velocity estimate is much smoother without being any more delayed.

Figure 3-2(a) demonstrates the quality of our EKF state estimates. We compared the EKF state estimates with ground-truth state estimates recorded by the Vicon motion capture system, and found that the estimates originating from the laser range scans match the ground-truth values closely in both position and velocity. Throughout the 1 minute flight, the average distance between the two position estimates was less that $1.5cm$, and the average velocity error was $0.015m/s$.



(a) (b)

Figure 3-2: (a) Comparison of the ground truth velocity(blue) with the estimate from the EKF before(red) and after(green) optimization. (b) The velocity error for the same trajectory.

3.3 Position Control

While the primary challenge for enabling indoor flight was obtaining sufficiently accurate real-time sensing, a smooth, stable and accurate controller is also necessary. While UAVs operating outdoors can generally afford to hover with an RMS error of several meters, indoor environments are much more constrained, requiring a hover precision on the order of tens of centimeters. While there has been a tremendous amount of work on performing system identification and designing controllers over the years [18, 43, 79, 20] , this section describes our relatively simple method for designing the vehicle controller, which was sufficient for our purposes.

The Ascending Technology quadrotors are already equipped with attitude stabilization, which uses an onboard IMU and processor to stabilize the MAV's pitch and roll [38]. This stabilizer tames the nastiest portions of the quadrotor's extremely fast, nonlinear, and unstable dynamics [43], allowing us to focus on stabilizing the remaining degrees of freedom in position and orientation. While the onboard controller simplifies the problem substantially, the quadrotor is still underdamped, requiring careful controller design to stabilize the vehicle.

The onboard controller takes 4 inputs:

$$\mathbf{u} = [u_\phi, u_\psi, u_t, u_\theta] \tag{3.19}$$

where u_ϕ and u_ψ denote the desired set points for the onboard PD control loops in pitch and roll respectively. Unlike these two control inputs, u_θ sets the desired rotational velocity in heading rather than specifying an absolute orientation. Finally u_t controls the desired baseline rotation rate for all four motors in the motor speed controller.

On a quadrotor, the vehicle acceleration is proportional to the pitch angle of the vehicle.

$$\ddot{x}^b \propto \cos(\phi) \tag{3.20}$$

$$\ddot{y}^b \propto \cos(\psi) \tag{3.21}$$

Therefore, by making a small angle assumption, the quadrotor's dynamics can be approximated as a simple 2^{nd} order linear system with the following equations:

$$\begin{aligned} \ddot{x}^b &= k_\phi u_\phi + b_\phi & \qquad \ddot{z} &= k_t u_t + b_t \\ \ddot{y}^b &= k_\psi u_\psi + b_\psi & \qquad \dot{\theta} &= k_\theta u_\theta + b_\theta \end{aligned} \tag{3.22}$$

where $\ddot{x}^b$ and $\ddot{y}^b$ are the resultant accelerations relative to the body coordinate frame, and k_* and b_* are model parameters that are functions of the underlying physical system such as mass and inertias.

To learn the parameters of this model we collect a log of input-output data by flying inside a Vicon [98] motion capture system while recording the pilot's control inputs. We first characterize the delay in each channel by computing the cross-correlation between the acceleration and the associated control input for a range of delays. We then align the accelerations with the control input, and use regression to compute the parameters of the control model. We use regularized least squares regression [74] to mitigate over-fitting. The learned model is able to predict the accelerations with reasonable accuracy as shown by the green line in figure 3.3. The figure shows a comparison between the predictions of our model and the measured accelerations.

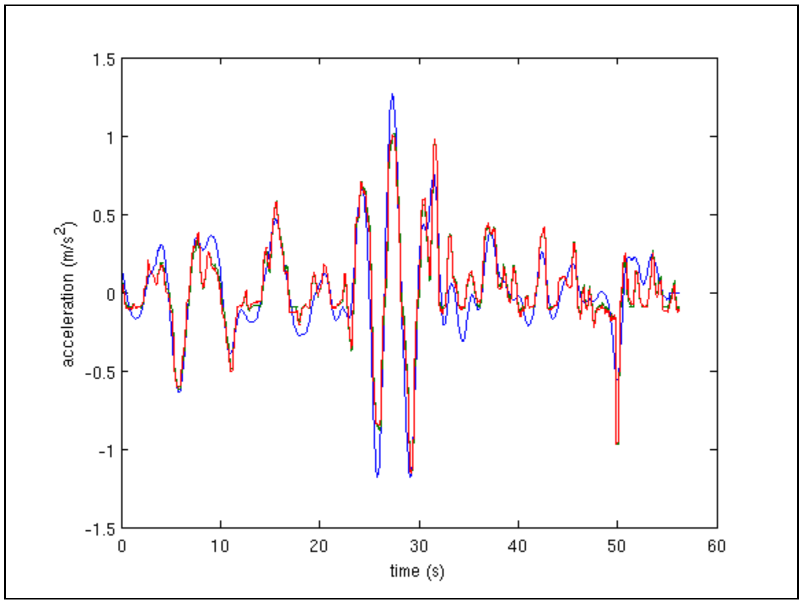


Figure 3-3: Comparison of the measured accelerations in the x-direction (blue) with the accelerations predicted by the dynamics models on a held out test data set. The predictions for other axes looks very similar. The green line shows the predictions from the dynamics model in equation 3.22, while the red line shows the predictions of the model with damping in equation 3.23.

In addition to the simple control model described in equation 3.22 we experimented with several other more complicated models such as

$$\begin{aligned} \ddot{x}^b &= k_\phi u_\phi + k_{\dot{x}}\dot{x} + b_\phi & \ddot{z} &= k_t u_t + k_{\dot{z}}\dot{z} + b_t \\ \ddot{y}^b &= k_\psi u_\psi + k_{\dot{y}}\dot{y} + b_\psi & \dot{\theta} &= k_\theta u_\theta + b_\theta \end{aligned} \tag{3.23}$$

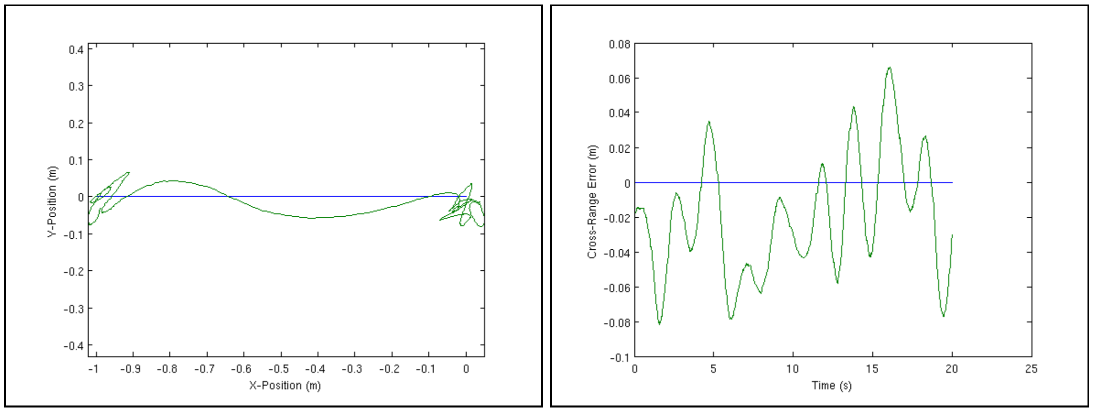
which included damping terms. When we fit data to this model, we found that the parameters on the damping terms were approximately zero. This confirmed our previous qualitative observation as pilots that the system is underdamped. The predictions made by this dynamics model are shown as the red line in Figure 3.3, where we can see that the predictions made by the two models are almost identical.

As mentioned in section 1.1 the underdamped nature of the system means that proportional control techniques alone will be insufficient to hover the vehicle, since any delay in the system will cause unstable oscillations. Fortunately, the state estimates computed by the data fusion module provide accurate, low delay velocity information that can be used by a derivative term in the feedback controller to add damping to the system.

We also experimented with other even more complicated models which included cross-coupling terms as well:

$$\begin{aligned} \ddot{x}^b &= \mathbf{k}_{\ddot{x}}\mathbf{u} + k_{\dot{x}}\dot{x} + b_{\ddot{x}} & \ddot{z} &= \mathbf{k}_{\ddot{z}}\mathbf{u} + k_{\dot{z}}\dot{z} + b_{\ddot{z}} \\ \ddot{y}^b &= \mathbf{k}_{\ddot{y}}\mathbf{u} + k_{\dot{y}}\dot{y} + b_{\ddot{y}} & \dot{\theta} &= \mathbf{k}_{\dot{\theta}}\mathbf{u} + b_{\dot{\theta}} \end{aligned} \tag{3.24}$$

where $\mathbf{k}_*$ is a vector of parameters, one for each control input. As in the damped model, we found that the additional terms seemed to have a negligible effect. We thought that these terms might have more complex and potentially nonlinear interactions with the system, so we experimented with using kernel-regression methods to fit the data. While the regularized least squares regression method that we employ helped reduce the amount of overfitting, the more complicated regression methods still overfit to the training data slightly, and performed worse than the simple linear models on held out test data. The results of these experiments are shown in table 3.1. Since none of the alternative dynamics models or regression methods significantly improved the fit of the data, we decided to use the simple 2^{nd} order dynamics model without damping or cross-coupling terms in the design of our feedback controller.



(a) (b)

Figure 3-4: Demonstration of the trajectory following performance with commanded position (blue) and actual position (green). The vehicle was commanded to move along the X-axis. (a) shows the position of the vehicle alongside the desired trajectory and (b) shows the cross-range error for this trajectory. The maximum cross-range deviation was $8cm$. The vehicle was flying autonomously with the state estimates generated by our system.

Regression Type	Dynamics Model	Training error	Testing error
linear	simple 2^{nd} order	0.086340	0.136492
linear	damping	0.084906	0.133933
linear	damping+cross	0.084570	0.131545
2^{nd} degree poly	simple 2^{nd} order	0.086305	0.138765
2^{nd} degree poly	damping	0.084708	0.137631
2^{nd} degree poly	damping+cross	0.083843	0.160786
3^{rd} degree poly	simple 2^{nd} order	0.086306	0.138572
3^{rd} degree poly	damping	0.084739	0.138129
3^{rd} degree poly	damping+cross	0.083762	0.163946

Table 3.1: Comparison of the performance of different dynamics models with parameters fit by different regression methods. We fit the parameters of the dynamics model using kernel regression with the kernel designated in the *Regression Type* column. The *simple* 2^{nd} *order* model corresponds to equation 3.22, while the *damping* model corresponds to equation 3.23, and *damping+cross* corresponds to equation 3.24

Once we obtained the parameters for the system dynamics in equation 3.22, we use the Matlab® linear quadratic regulator (LQR) toolbox to find appropriate feedback gains for the dynamics model. Despite the contrast between the seemingly complex dynamics of the vehicle and the model's apparent simplicity, this controller achieves a stable, precise hover. The controller initially had a fairly large steady state error, which was mitigated by adding a low gain integral feedback term around the LQR controller. The final controller is able to hover the quadrotor, and accurately follow trajectories with under $6cm$ cross-range error. An example of the vehicle following a simple trajectory, along with the cross-range error is shown in Figure 3.3.

3.4 SLAM

The odometry and data fusion EKF combine to provide locally accurate state estimates which enable the vehicle to hover and move around room sized environments, however, as the vehicle moves around larger areas, small errors in the odometry will accumulate, resulting in position drift over time. With the MAVs fast dynamics canceled out by lower layers we can leverage SLAM algorithms originally developed for other platforms, to close loops and create globally consistent maps. The SLAM process could be integrated directly inside of the EKF data fusion filter described above, however, this would likely add a significant amount of computational delay to the real-time loop since the filter state would expand to include the positions of environmental features. Instead we chose to keep the SLAM process separate, and have it provide corrections for the real-time position estimates. This allows the SLAM algorithm to take much more time to integrate information than would be possible if it was in the real-time loop. In addition, the system is modular, allowing different SLAM algorithms to be tested without modifying the rest of the system.

While there has been a tremendous amount of research on SLAM algorithms, the vast majority of the algorithms have focused on building $2D$ maps. There has been success using $3D$ laser scanners for $3D$ SLAM [21, 68, 39], but these sensors are still outside the realm of possibility for indoor MAVs. More recently, several groups have begun to achieve success with using either monocular [28, 56] or stereo cameras [72] for performing SLAM. These systems appear to work quite well in practice and warrant further consideration however none of the publicly available implementations scale to large enough environments to be of use in our situation. Perhaps the most exciting solution to the loop closure problem has recently been proposed by Cummins et al who use a purely appearance based method to close loops in large scale environments [26]. In recent work, they have combined this approach with a high performance visual odometry system to perform large scale real-time topometric SLAM with very impressive results.

Despite the recent progress in visual SLAM, we decided not to tackle the problem of building our own $3D$ SLAM system. Instead we made use of the publicly available implementation of the GMapping [34] algorithm that is available in the OpenSlam repository [71], which performs slam in $2D$. Despite the fact that the MAV operates in the full $3D$ environment, the algorithm works surprisingly well and serves as a proof of concept for implementing SLAM on a MAV. In the future, we hope to replace this module with a more capable fully $3D$ SLAM approach.

GMapping is an efficient Rao-Blackwellized particle filter which learns grid maps from laser range data. We chose it due to its outstanding accuracy, real-time performance, and its ability to handle changes to the map that occur due to changing height and attitude.

While the algorithm worked reasonably well out of the box, we made modifications that improved its performance when used in $3D$ environments on a MAV. The motion model for the particles in the GMapping algorithm was based on a standard motion model for ground robots with wheel odometry. However, since we use the odometry estimates computed by the laser scan-matching module, we modified the motion model to propagate the particles using the uncertainties computed by the odometry module rather than using fixed noise terms.

In addition to the motion model, we modified the map representation so that the map updates rapidly in response to changes in height. The algorithm computes the probability that each grid cell is occupied or free based on the number of times a laser beam reflects off, or passes through the cell. If a particular cell has been hit many times, the algorithm places a very high confidence in the fact that the cell is occupied. However, if the MAV changes heights, and the cell becomes part of free space, this confidence is no longer warranted. With the original map representation, the laser must pass through the cell at least as many times as it was hit before the algorithm will be convinced that the cell is actually now free, resulting in a very slow adaptation of the map. As a result, we modified the map representation to cap the maximum confidence for each grid cell, allowing it to change from occupied to free (or visa-versa) more rapidly.

With these modifications, we are able to create large scale maps of the environment which will be shown in chapter 4. GMapping usually takes 1 to 2 seconds to process incoming laser scans which allows it to be run online, but is not suitable to be directly incorporated into the real-time control loop. Instead the GMapping algorithm periodically sends position corrections to the data fusion EKF to correct the drift in the position estimates due to errors in the odometry. Since the position corrections are delayed significantly from when the measurement upon which they were based was taken, we must account for this delay when we incorporate the correction. This is done by retroactively modifying the appropriate position estimate in the state history. All future odometry estimates are then recomputed from this corrected position, resulting in globally consistent state estimates. By incorporating the SLAM corrections after the fact, we allow the state estimates to be processed with low enough delay to control the MAV, while still incorporating the information from SLAM to ensure drift free position estimates.

3.5 Planning and Exploration

In addition to allowing us to compute globally consistent state estimates, the map generated by the SLAM algorithm is used for planning autonomous actions for the vehicle. To achieve full autonomy, we require a high-level planner that enables the MAV to either explore environments autonomously. While exploration has been well-researched in ground robotics, differences between air and ground vehicles, as discussed in section 1.1, require different considerations when deciding where to go next. In particular, the need to constantly provide control signals to the MAV means that while we seek to explore the environment, we must ensure that the MAV always remains well-localized. Our algorithm trades off the speed with which our robot completes coverage of the environment with safety. For example when confronted with a large open area, an exploration algorithm for a ground robot would drive directly into the open area, thereby uncovering the maximum number of unexplored cells whereas our algorithm moves the vehicle to positions in which known environmental features are visible as well as unexplored areas. These positions allow the vehicle to localize itself, however they uncover less of the unexplored environment.

We use a modified definition of frontiers, first proposed in [100], to choose possible positions in free space where the MAV should fly to next such that it explores previously unexplored regions in the environment. In [100], free cells that are adjacent to unknown cells are grouped into frontier regions as possible goals for the robot. We use a similar method to identify frontier regions, however, for each of these frontier regions, we seek to find a frontier pose that maximizes both the amount of unexplored space that is expected to be explored and the ability of the MAV to localize itself, which we define below.

The first step in our exploration algorithm is to identify candidate frontier regions. Frontier regions are areas in the map where there is a direct transition between free and unexplored cells. Since the walls in occupancy maps such as those generated by GMapping may have small gaps, the set of regions is then filtered to remove spurious frontiers. The algorithm must then identify the pose within these frontier regions that provides the best tradeoff between localization ability, and uncovered area. Searching over all poses in the frontier regions is too slow to allow the algorithm to run online, so frontier poses are sampled in each region. For each sample, two metrics are used to calculate a weight associated with each sample. First, the amount of unexplored space that the MAV will explore can be calculated by simulating the laser sensor data that the MAV is expected to obtain at the sampled pose, given the latest map. By extracting the number of grid cells within the laser's field-of-view that are currently unexplored and dividing by the maximum number of grid cells covered by a laser range scan, we get a normalized weight, $\mathcal{I}_{\mathcal{UR}}(x)$ in the range of $[0, 1]$ for the amount of unexplored information that the MAV is expected to observe.

Using this metric alone will result in frontier points that are at the extreme borders of the map facing the unexplored region, since such a pose will maximize the number of grid cells in the laser's field-of-view that are unexplored. Unfortunately, this pose provides no guarantees that the MAV will be able to localize itself, since the unknown environment may not contain enough structure for the relative position estimation algorithms to match against. In the extreme case, the MAV could be facing an open space where the nearest walls are beyond the maximum range of the laser scanner, giving the MAV no information with which to localize itself.

We therefore add an additional "Sensor Uncertainty" metric, first coined in [89]. Sensor uncertainty is used to quantify the MAV's ability to localize itself at different positions in the map. A sensor uncertainty field maps locations x in the map to expected information gain, $x \rightarrow \mathcal{I}_{\mathcal{SU}}(x)$, by calculating the difference in entropy of the prior and posterior distribution

$$\mathcal{I}_{\mathcal{SU}}(x) = H(p(x)) - H(p(x|z)) \tag{3.25}$$

where entropy is

$$H(p(x)) = -\int p(x) \log p(x) dx \tag{3.26}$$

Shown in [41], the measure of information gain for laser data is typically insensitive to the choice of prior. We therefore use a constant prior $p(x) = \Sigma_0$ such that $H(p(x)) = C$, as

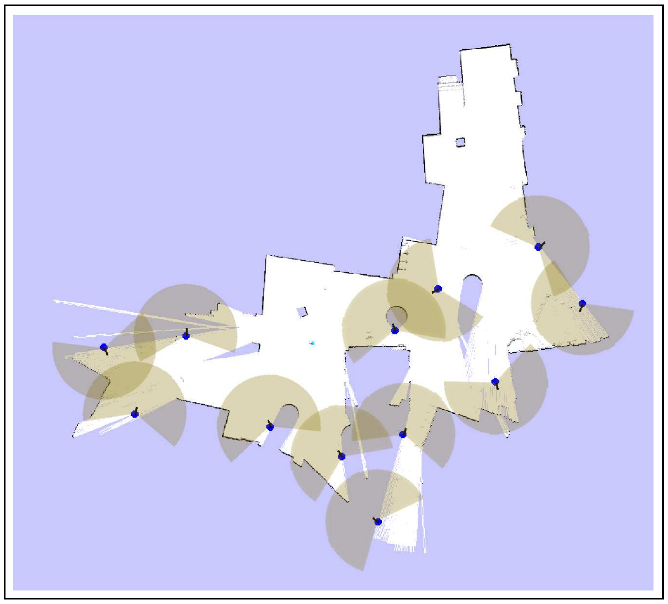


Figure 3-5: The blue pointers indicate frontiers that allow the MAV to explore and self-localize simultaneously. The laser's field-of-view at those frontiers is drawn in brown. Notice that at the edges of free space, the chosen frontiers position the vehicle such that the expected laser scan spans both unexplored regions for exploration and unique obstacles for localization.

well as Bayes' rule to compute $p(x|z) = p(z|x) \cdot p(x)$, such that

$$\mathcal{I}_{\mathcal{SU}}(x) = C - H(p(z|x))\Sigma_0 \tag{3.27}$$

We compute the entropy of $p(z|x)$ by deterministically extracting a set of sigma points [50], or samples along the main axes of the current covariance estimate, and observing how they are transformed when they are passed through the measurement function. For each sample, we simulate the sensor measurements and find the probability of observing the sensor measurement at each of the sigma points. The lower the probability of observation at the neighboring sigma points, the smaller the entropy of the posterior distribution, and therefore the greater the information gain. Locations with high information gain correspond to locations that generate sensor measurements that we expect to maximize the localization accuracy of the vehicle. After normalizing this with the prior entropy, $\mathcal{I}_{\mathcal{SU}}(x)$ is also a weight that lies in the range of $[0, 1]$.

Using these two weights, we are able to find frontiers that maximize both the exploration and localization capabilities of the MAV. In each frontier region, we sample a set of candidate poses, and accept as the goal point for that region, the sample that maximizes the weighted sum of the two information metrics, such that $\mathcal{I}(x) = \mathcal{I}_{\mathcal{UR}}(x) + \mathcal{I}_{\mathcal{SU}}(x)$. Figure 3-5 shows the frontier points generated accordingly, where points are chosen such that the expected laser scan will both uncover unexplored regions and observe known obstacles, enabling the MAV to simultaneously explore and localize.

To achieve autonomous exploration of an unknown environment, the planner uses the nearest frontier as its goal and computes a path using the path dynamic programming based path planner in the Carmen Robot Navigation Toolkit [2]. The frontier extraction modules run fast enough that they are able to generate plans, and re-plan online as the vehicle moves through the environment and the map is updated.

3.6 Obstacle Avoidance

The final necessary piece of our system is the obstacle avoidance module. While the planner plans paths that should keep the vehicle a safe distance from any obstacles, things do not always work out as planned. For example, the map updates and plans take several seconds to generate, during which time new obstacles can appear due to height changes. In addition, while the integral term in the controller should eliminate biases in the vehicle position, it is not always perfect. As a result the vehicle will sometimes stray dangerously close to obstacles. Even the slightest bump can result in crashes and damage the MAV, so it is critical to ensure that it always maintains a safe distance from any obstacles.

Since the map updates can have high latency, resulting in a map that is out of date, we use the raw point cloud P generated by the laser and stereo in the body-centered coordinate to perform obstacle avoidance. By computing the obstacle locations in the body frame, we allow the obstacle avoidance module to be independent of the rest of the modules in the system. When the obstacle avoidance module detects that it is too close to an obstacle it sends a message directly to the control module which shifts the entire trajectory that the vehicle is trying to follow. The obstacle avoidance module does not attempt to reason about the current goal of the robot, instead relying on the planning module to re-plan if necessary to get the robot back on course.

To determine the direction and magnitude of the correction required, we compute a vector which is the weighted sum of the repulsion forces for all the points that are too close. More concretely, let $\mathbf{p} \in P$ be the location of a point in the body coordinate frame, and let d be the maximum distance for which we want to apply a correction; if P_C, the set of points within d of the vehicle is not empty, we compute the correction vector $\mathbf{v}$ as

$$\mathbf{v} = \sum_{\mathbf{p}_i \in P_C} \frac{d}{||\mathbf{p}_i||^2}(-\mathbf{p_i}) \tag{3.28}$$

This vector is scaled by a constant factor that depends on the rate at which we send corrections and the maximum expected velocity of the vehicle such that the correction is applied gradually and does not overwhelm the position controller.

Chapter 4

Experiments

In this chapter, we present results demonstrating the capability of our system. We integrated the suite of technologies that were described in previous chapters to perform fully autonomous navigation and exploration in a variety of unstructured and unknown indoor environments. The experiments were performed using 3 different hardware configurations with different sensor suites. We first describe the hardware used before presenting the results from flight tests. To get a full picture of our system in action, we suggest that the reader also view the videos taken of these experiments available at:
`http://groups.csail.mit.edu/rrg/videos.html`.

4.1 Hardware

We addressed the problem of autonomous indoor flight as primarily a software challenge, focusing on algorithms rather than exotic hardware. To that end, we used consumer off-the-shelf hardware throughout the system.

Our system was primarily built around the Hummingbird quadrotor helicopter designed by Ascending Technologies GmBH [1]. This vehicle provided us with an extremely robust, stable, and safe platform with which to experiment. This robustness was critical for the development of the system, as any attempts at performing indoor flight will inevitably result in crashes. With the Hummingbird quadrotor, even when we did experience crashes, the damage was usually minimal, only requiring replacement of the soft plastic rotors.

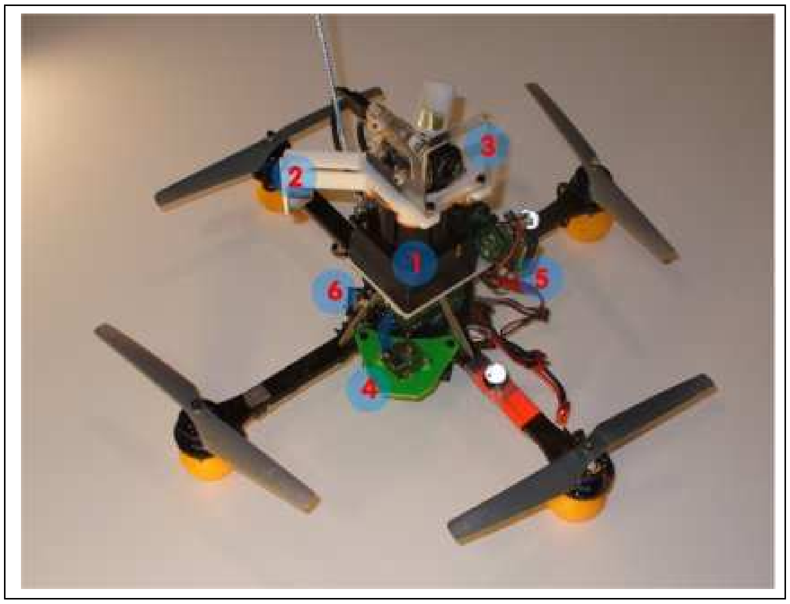


Figure 4-1: Our laser-equipped quadrotor helicopter. Sensing and computation components include a Hokuyo Laser Range-finder (1), laser-deflecting mirrors for altitude (2), a monocular camera (3), an IMU (4),

The Hummingbird is able to carry roughly $250g$ of payload, which allowed us to carry either a laser range scanner, or a stereo camera rig, but not both.

4.1.1 Laser Configuration

The first vehicle that we achieved autonomous flight with, shown in Figure 4-1, was equipped with a Gumstix [37] microcomputer and a lightweight Hokuyo [44] UTM-30LX laser range-finder. The laser range-finder provides a $270°$ field-of-view at $40Hz$, up to an effective range of $30m$. We deflect some of the laser beams downwards using a right angle mirror to estimate the vehicle's height, while the rest are used for localization. In earlier experiments we also used the shorter range (5.6m) Hokuyo URG-04LX, which worked well for small environments, but would not allow the MAV to localize in larger environments. All computation is done offboard with the Gumstix used solely to obtain sensor data and transmit data and commands between the vehicle and the ground-station over 802.11g. A lightweight webcam was mounted as well to obtain first person video of the flight, however it was not useful for localization due to the low frame rates and distortion caused by the rolling shutter.

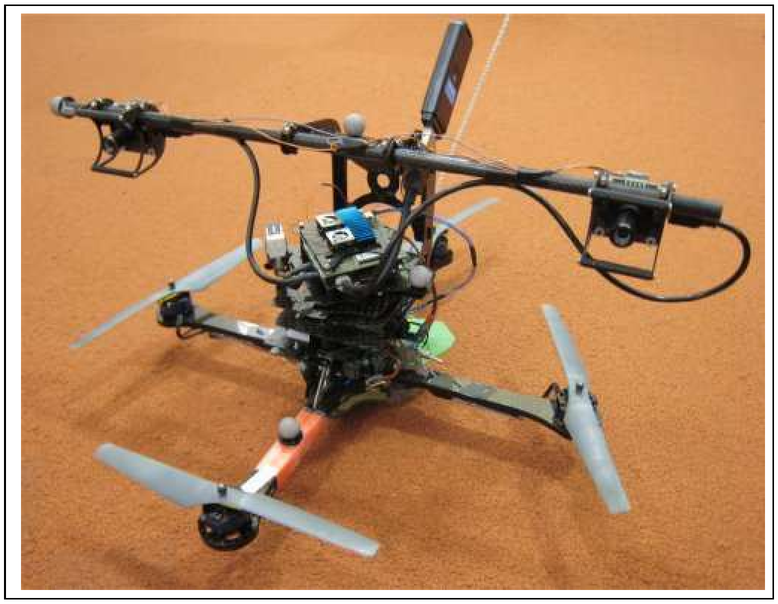


Figure 4-2: Our quadrotor helicopter equipped with a stereo camera rig, Intel Atom® based onboard computer and USB Wifi dongle.

4.1.2 Stereo Configuration

The second vehicle to achieve autonomous flight, shown in Figure 4-2, was equipped with a synchronized pair of lightweight uEye LE WVGA monochrome USB cameras [88] and a more powerful embedded computer. We chose to use monochrome cameras due to their superior light sensitivity and lower data rates. The fast motion of the vehicle mandates the use of cameras with global shutters, which eliminates the possibility of using cheap webcam cameras such as the one mounted on the laser vehicle. Interfacing with the new cameras required an onboard computer with full high-speed USB2.0 capability, which meant that we had to search for a replacement for the Gumstix microcomputer.

Intel recently introduced its Atom® platform for mobile devices, an ideal choice for our application. Compared to other platforms, such as those equipped with Via or Arm processors, the Atom platform has some major advantages. Since the processor uses the x86 architecture, standard operating systems and even more important, standard drivers for devices like cameras, and WiFi dongles could be used. This became important especially in cases where only binary drivers are available. Furthermore, Intel's performance libraries like IPP (Integrated Performance Primitives, [23]) and MKL (Math Kernel Library, [24]) are available for the Atom processor which significantly increases the speed of tasks such

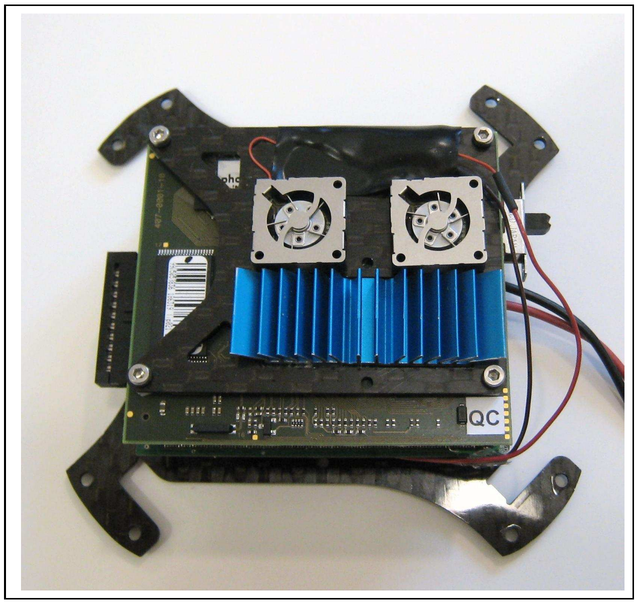


Figure 4-3: Our Intel Atom® based flight computer.

as image compression or large matrix operations. For example, the use of IPP for JPEG compression yielded an order of magnitude speedup.

Lippert Embedded [33] recently packaged an Intel Atom processor into a Computer-On-Module (COM) form factor. This device has dimensions of only $58 \times 65mm$, weighs $26g$, but includes a $1.6Ghz$ Atom processor, chipset, and $1GB$ of RAM. All I/O and interfaces such as USB are only available on a high density 220-pin connector which required us to design a carrier-board to provide access to the necessary interfaces. Altogether with the carrier board, the complete computer weighs a total of $60g$. The complete computer is shown in Figure 4-3. We attached a standard Linksys 802.11n USB wifi dongle to provide a high bandwidth link to the ground-station.

In addition to the improved performance of the new computer over the Gumstix, perhaps the most important change was that since it ran the same operating system and environment as the base-station, it became easy to move processes that had been run on the base-station to run onboard the quadrotor, without having to deal with cross-compilation or library availability issues. Moving the position controller and data fusion module onboard reduced the delay, and made the vehicle less susceptible to failures due to the wireless connection.

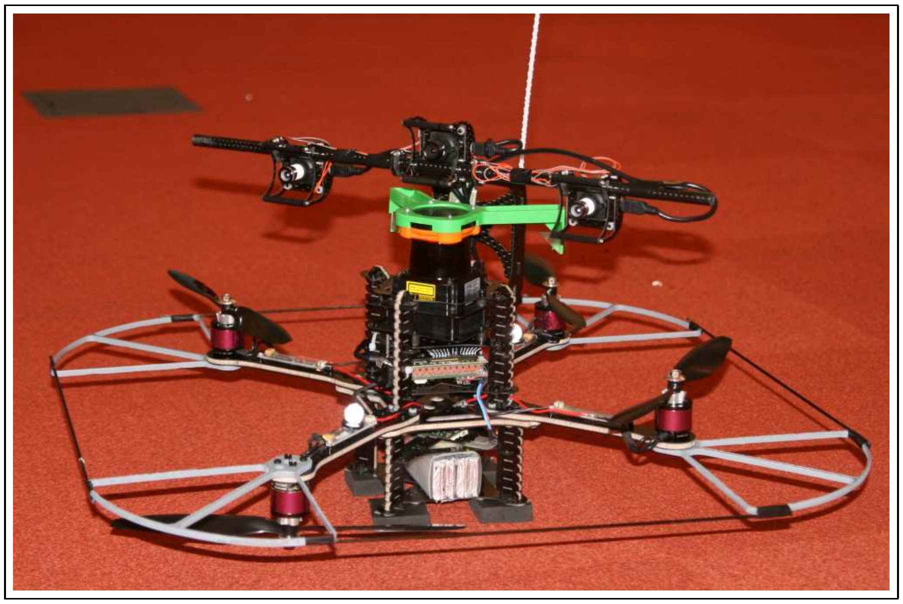


Figure 4-4: The Asctec Pelican, outfitted with both the laser range-finder and stereo camera. A third color camera is mounted in the center.

4.1.3 Combined Configuration

While the two platforms described above were very capable, both the laser and stereo sensors have environments where they are unable to localize the vehicle. As a result, we sought to combine the two sensing modalities on a single vehicle. Unfortunately, as it was we were pushing the payload limits of the Hummingbird quadrotor, so a new vehicle was needed to be able to support the larger payloads. In collaboration with Ascending Technologies we built a larger quadrotor named the Asctec Pelican, shown in Figure 4-4 that is able to carry the $500g$ payload containing the stereo pair, laser scanner, computer, and all supporting cables, connectors, and mounting hardware. The new vehicle uses larger 10-inch rotors and more powerful motors compared to the hummingbird. The larger rotors increase the maximal dimension of the quadrotor from $55cm$ to $75cm$.

4.2 Flight Tests

We performed a number of experiments with each of the sensing modalities described in this thesis, testing the performance of our systems in different real world environments. For each sensor system, we thoroughly tested each of the modules by flying inside the motion capture studio described in section 3.2.4 before venturing into real world environments. These tests ensured that the vehicle is able to accurately fly in small environment where we have access to the ground truth state of the vehicle for debugging, and as a "safety net" in case something goes wrong. Once we were convinced that the system was working sufficiently well, we moved to fly in other larger, and less constrained environments.

4.2.1 Laser Only

Autonomous navigation in open lobbies To test the large scale navigation capabilities of our laser based system, we flew the vehicle around the first floor of MIT's Stata Center. The vehicle was not given a prior map of the environment, and flew autonomously using only sensors onboard the MAV. In this experiment, the vehicle was guided by a human operator clicking high-level goals in the map that was being built in real-time, after which the planner plans the best path to the goal. The vehicle was able to localize itself and fly stably throughout the environment, and Figure 4.2.1 shows the final map generated by the SLAM algorithm at the end of the experiment. As can be seen from the map, the Stata Center has a very free-form floor plan which would prevent the use of specific environmental assumptions such as straight hallways, as was done in [19] and [49]. Despite these challenges, our system worked quite well, allowing the vehicle to fly until the battery was exhausted. During the 8 minute flight, the vehicle flew a distance of $208.6m$.

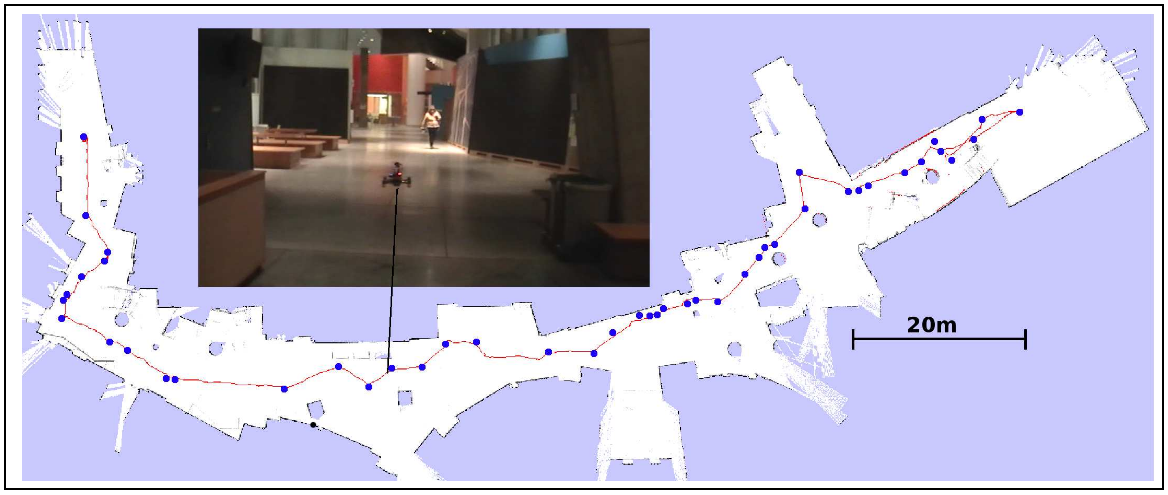
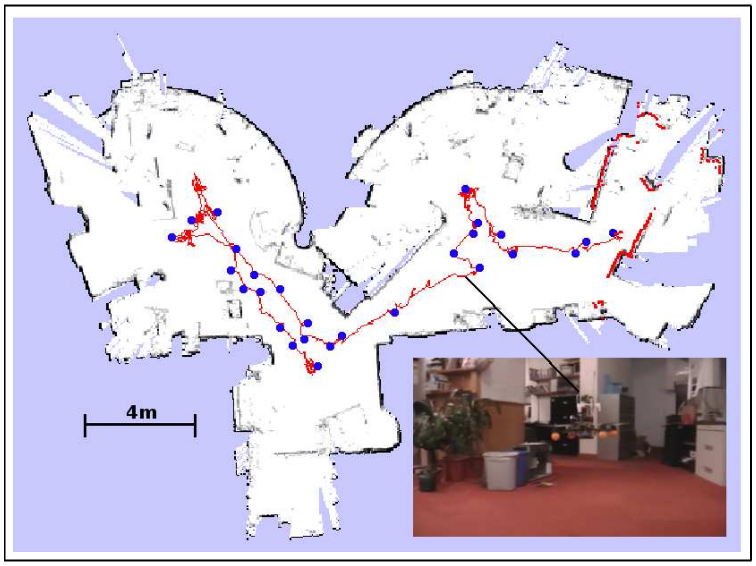
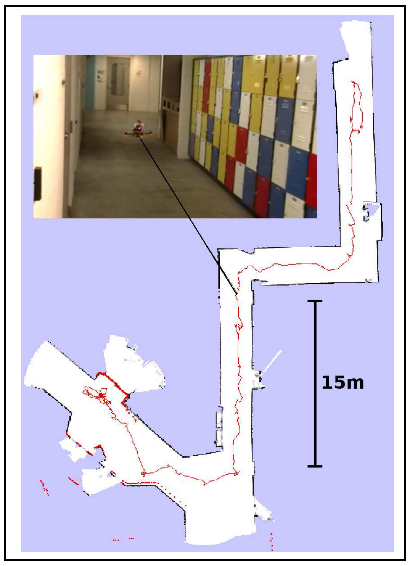


Figure 4-5: Map of the first floor of MIT's Stata center constructed by the vehicle during autonomous flight. The blue dots indicate goal points selected by the user.

Autonomous navigation in cluttered environments While unstructured, the first floor is a wide open environment without much $3D$ structure, which allows our $2D$ map assumption to hold fairly well. To verify that the laser system is robust against significant amounts of $3D$ structure, we next tested our system by flying through the cluttered lab space shown in the insert of Figure 4-6(a), operating close to the ground. At this height, chairs, desks, robots, plants, and other objects in the area cause the 2D cross-sectional scan obtained by the laser range-finder to vary dramatically with changes in height, pitch, and roll. The resultant SLAM map of the environment is shown in Figure 4-6(a). The gray features littered within the otherwise free space denote the objects that clutter the environment and are occasionally sensed by the laser range-finder. Despite the cluttered environment, our vehicle was able to localize itself and maintain a stable flight for 6 minutes over a distance of $44.6m$, a feat that would not have been possible with a static map assumption.



(a) Map of MIT Stata Center, 3rd Floor.



(b) Map of MIT Stata Center, basement.

Figure 4-6: (a) Map of a cluttered lab space with significant 3D structure, the blue dots are the high level goals selected by the user. (b) Map of constrained office hallway generated while performing completely autonomous exploration.

Autonomous exploration in office hallways Finally, to demonstrate fully autonomous operation of the vehicle, we closed the loop with the exploration algorithms described in section 3.5. The MAV was tasked to explore the hallway environment shown in Figure 4-6(b). Once the MAV took off it was completely autonomous, with no human control over the MAV's actions as it explored the unknown environment. The MAV continuously searched for new frontier goals and generated paths to these areas of new information. Figure 4-6(b) shows the map built from 7 minute autonomous flight, after traveling a distance of $75.8m$.

4.2.2 Stereo Only

Since we did not develop a visual slam solution to perform mapping, or close loops and correct for drift in the visual odometry algorithm, we were not able to test the large scale autonomous flight capabilities of the vehicle with stereo only. However, we were able to verify that we can close the loop and stabilize the vehicle using only vision and inertial sensing in a number of environments. In fact, these experiments verified the generality of our multi-layered sensing architecture as *no modifications* were required to switch between using the laser scan-matching and the visual odometry. The lower data rates, and slightly higher noise of the visual odometry made the vehicle oscillate more, but achieving closed loop control on the first try was very gratifying. Figure 4-7 shows the comparison of the position and velocity estimates to the ground truth state estimates from motion capture.

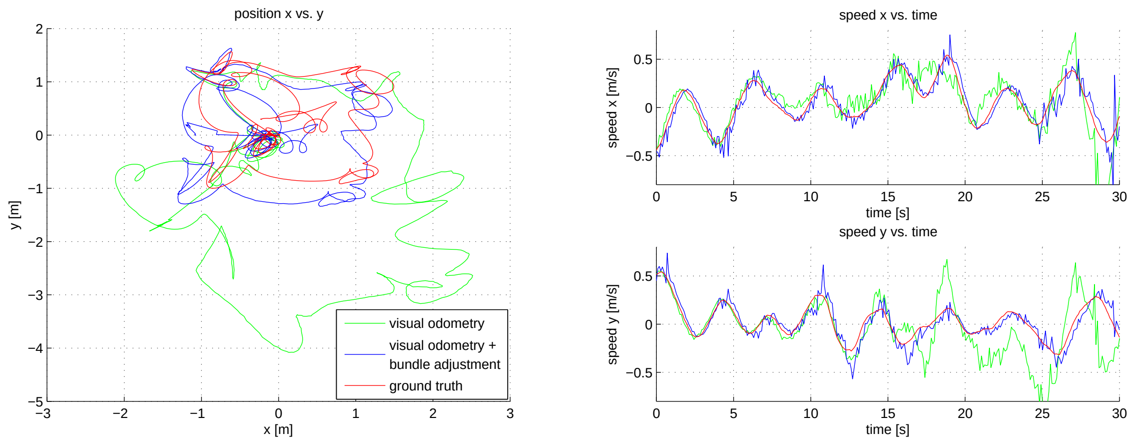


Figure 4-7: Position and speed over 1200 frames estimated by simple visual odometry from frame to frame (green) and by optimization with bundle adjustment (blue) compared to ground truth (red). The vehicle was flying with position control based on the estimates from bundle adjustment

In addition to these quantitative experiments performed in the controlled motion capture environment, we also flew the vehicle autonomously in a variety of other environments. Figure 4-8 shows the vehicle operating under vision based control in several of the environments in which we tested.

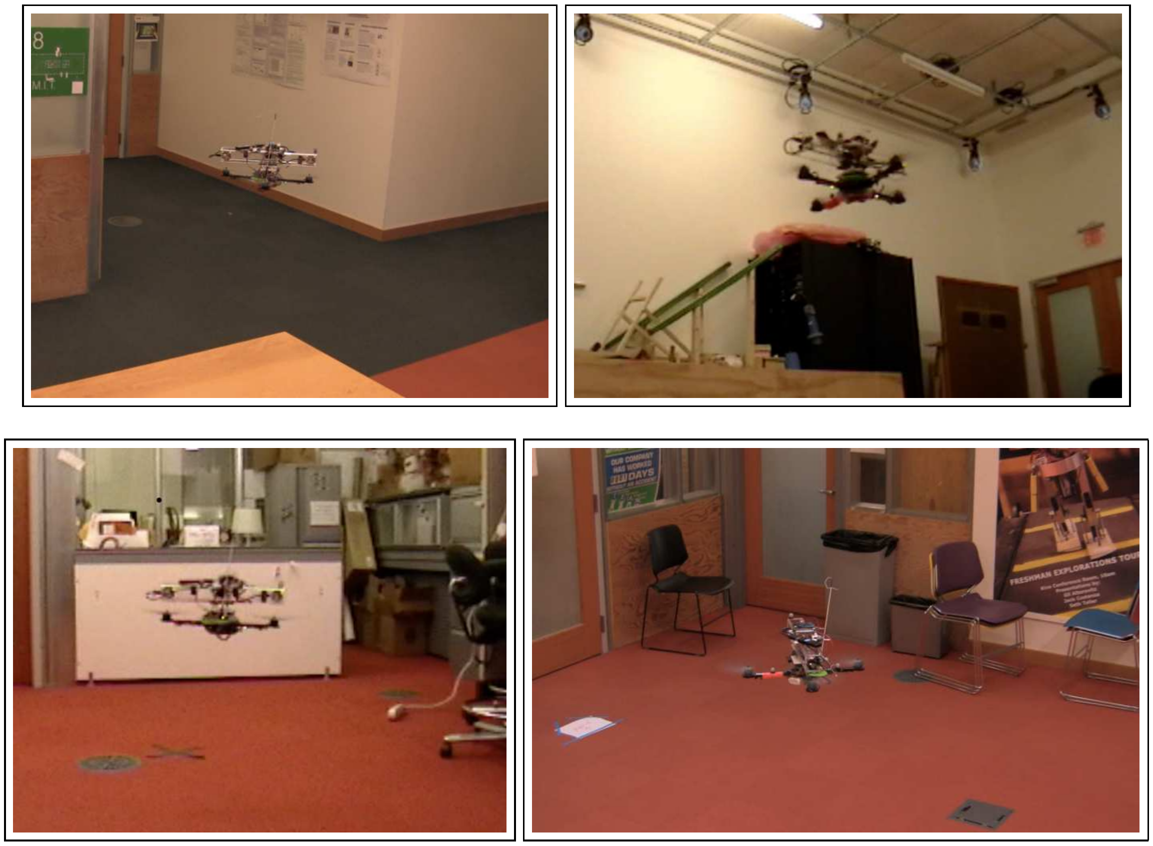


Figure 4-8: Pictures of the quadrotor flying autonomously using stereo vision only in a variety of environments.

4.2.3 Laser and Stereo

The system presented in this thesis was used in our winning entry as Team MIT-Ascending Technologies in the 2009 International Aerial Robotics competition (IARC), hosted by the Association of Unmanned Vehicle Systems International (AUVSI). Our team used the Asctec Pelican quadrotor described above and employed the complete system presented in this thesis for state estimation, control and mapping. We developed an additional set of new modules that performed tasks specific to the competition. These modules fit into the system at the top of the system hierarchy, and therefore did not need to be concerned with any of the state-estimation or control issues.

The 5th mission of the International Aerial Robotics Competition was founded on the premise of advancing the state-of-the-art in indoor aerial robotics. After organizing 4 outdoor missions over a span of nearly two decades, the 5^{th} mission was started in 2009. The 5^{th} mission was based around a disaster recovery scenario, where a MAV was to be used to help diagnose a fault in a melting down nuclear power plant. The MAV was given the task of entering the plant through an open window and searching the plant to find a control panel that contained critical information for the team of technicians trying to diagnose the fault in the nuclear reactor. Our vehicle was able to complete the mission in its entirety thereby winning the competition. In the 19 year history of the IARC, this was the first time that a team managed to complete the mission in its first year.

The actual competition was held in a gym, where artificial walls were set up to simulate the interior of a powerplant. An overhead view of the 30x$15m$ competition arena is shown in Figure 4.2.3. Prior to the mission, the arena layout was unknown. The specific mission entailed the following tasks:

1. Takeoff roughly $3m$ from the opening of the arena.

2. Identify and fly into the arena through a 1x$1m$ window opening.

3. Explore the unknown environment and search for the control panel. The correct gauge on the control panel was designated by a steady blue LED.

4. Autonomously designate and send imagery of the gauge to the judges.

The mission had to be performed completely autonomously in under 10 minutes. Once the operator told the vehicle to start, no human input was allowed until the mission was complete. Each team was given 4 attempts at completing the mission

One of the most challenging parts of the mission was flying through the window to enter the arena. The window opening was only $25cm$ wider than the width of the Pelican quadrotor. Assuming that we were able to perfectly identify the center of the window, the dimensions were such that we could only afford a $12cm$ maximum deviation from the desired straight line trajectory through the window. However, this is well within the 6cm error of our controller described in section 3.3. Any offset in the position controller, or error in the localization of the window center could easily result in a catastrophic crash.

To detect the window, we decided to use the laser scanner to generate a $3D$ point cloud of the window from which we could extract the location of the window. The laser scanner only measures the $2D$ plane around the vehicle, so to generate the $3D$ point cloud of the window, we used the vehicle to perform a vertical "sweep" with the laser. With the accurate position estimates generated by our state estimation system, we were able to register the set of laser scans into the dense $3D$ point cloud shown in Figure 4-9(b). The $3D$ point cloud allowed us to identify the window location by explicitly searching for a region in the data which had a roughly 1x$1m$ rectangular hole in an otherwise connected flat surface. The resulting window location is designated by the pink lines in Figure 4-9(b). Once we detected the location of the window, the planning module instructed the vehicle to fly through a set of waypoints that took the vehicle through the center of the window along a path that was perpendicular to the window. A photo of the vehicle executing this path is shown in Figure 4-9(a).

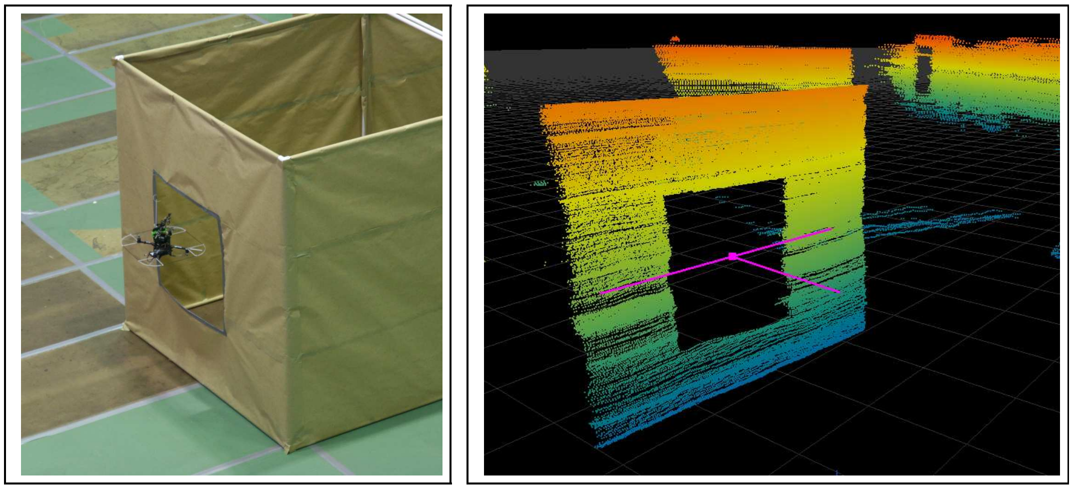

(a) (b)

Figure 4-9: (a) A photo of our MAV autonomously flying through the 1x$1m$ window to enter the competition arena. (b) A $3D$ rendering of the point cloud used to detect the window. The detected position of the window is designated by the pink lines.

The window detection, and window entry routine described above performed flawlessly at the IARC competition. We were able to successfully detect and fly through the window over ten times during the course of the competition. This performance is a clear demonstration of the accuracy and precision of the state estimates generated by our system. If the state estimates had more noise or delay, it would not be possible to perform precision maneuvers such as flying through the narrow window. The quality of the $3D$ point cloud is another clear demonstration of the accuracy of our state estimates. While we do not have ground truth information for the $3D$ shape of the environment, we can visually inspect the point cloud and see that the walls are not significantly distorted.

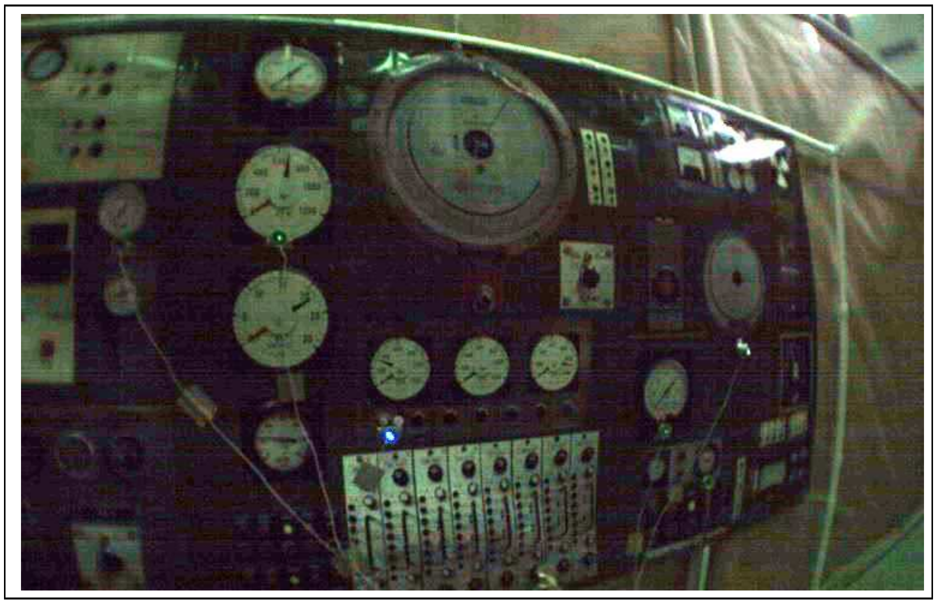


Figure 4-10: Photo of the IARC control panel. This is one of the frames from the video stream sent to the judges.

Once through the window, the vehicle switched into exploration mode, and began searching the environment for the control panel. The vehicle searched the building using an exploration strategy based on the one described in section 3.5. During the first three attempts, the vehicle explored much of the environment, however we were unable to reach the room with the control panel. On the way to the room, there were several doors which were exactly $1m$ wide. We had not anticipated that the vehicle would have to go through openings that small other than the window into the arena, so we had set the “safety regions” for the obstacle avoidance and path planning modules such that they thought the doorways were impassable. After adjusting parameters, on our fourth attempt we managed to enter the control room and take the picture of the control panel shown in 4-10, which completed the mission. The path followed by the vehicle during the successful attempt is shown on top of the constructed map in Figure 4-12. We completed the mission in just 4.5 minutes, however, we were lucky that the exploration strategy randomly decided to go left at the junction near the end of the initial hallway.

We were very happy with the performance of our vehicle, and the system presented in this thesis. The system provided a very robust and stable platform on top of which we were able to build the necessary additional components that carried out the mission. All of these components were linked into the system at the top level of the hierarchy, which meant that the modules could take as much time as they needed, and did not have to be concerned with stabilizing the vehicle or any of the other real-time concerns that make developing algorithms for MAVs particularly challenging.

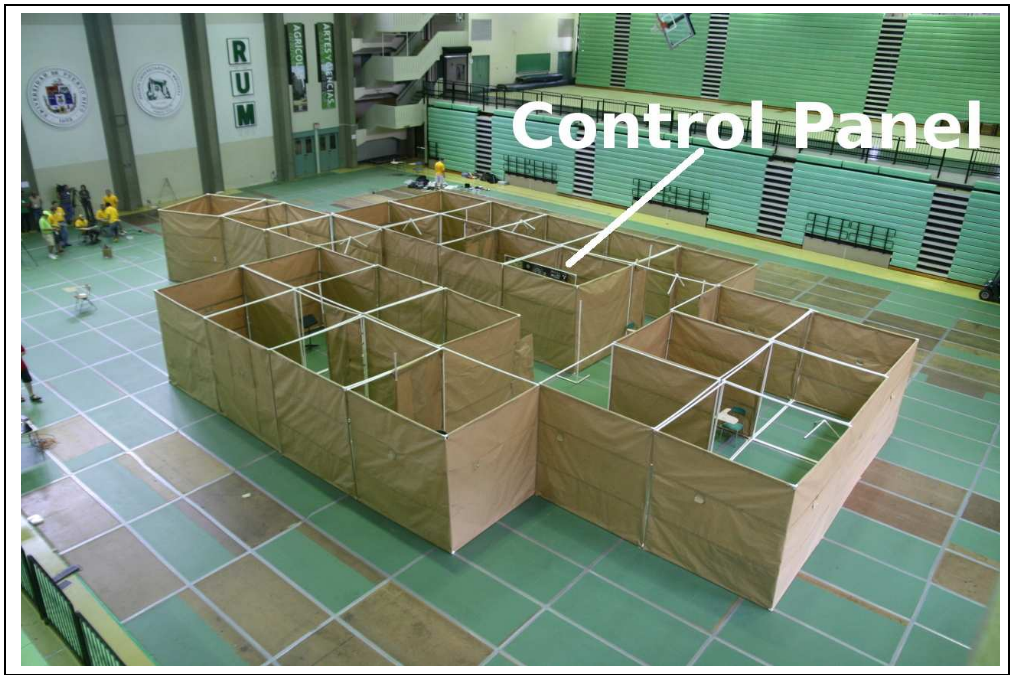


Figure 4-11: An overhead view of the competition arena used for the 2009 AUVSI International Aerial Robotics Competition, 5^{th} mission.

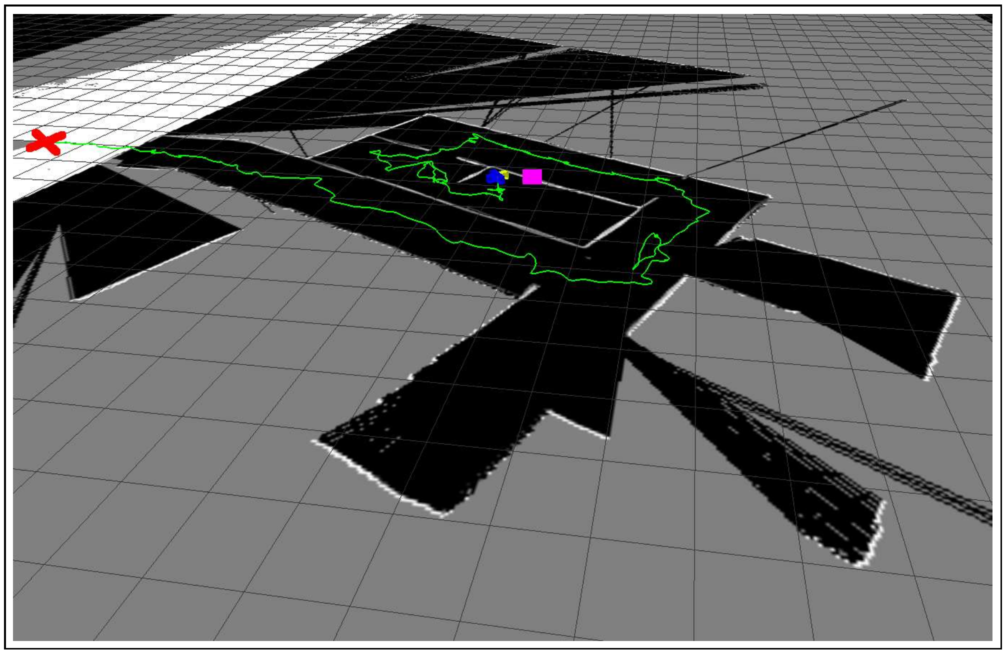


Figure 4-12: The map built by the vehicle as it explored the IARC arena searching for the control panel. White areas are obstacles (or no-fly zones), black is free space, and gray is unexplored. Starting from the red “X” on the left, the green line shows the path taken by the vehicle. The blue and yellow cross is the final position of the vehicle. The magenta square in front of the vehicle is the location of the autonomously designated control panel.

Chapter 5

Dense 3D Environment Modeling

In this chapter I discuss initial progress towards developing a more complete $3D$ model of the environment around the MAV. I first present the motivation for tackling this problem and related work, before describing our approach.

While the $2D$ model used throughout the system described in previous chapters allowed us to successfully stabilize and operate the vehicle in $3D$, it did not allow us to *plan* actions in $3D$. For example, when the vehicle encountered an obstacle such as a table, while it was perfectly capable of flying over the table if instructed to do so by a human operator (who had a $3D$ model in their head), with a $2D$ world model, the vehicles path planner could only plan a path around the table, rather than over it. We are particularly interested in tackling problems that enable planning, sensing and control in the full $3D$ environment, a domain that is unique to autonomous air vehicles. Without this capability, operating in an environment such as the one shown in figure 5-1 would likely be impossible.

To plan actions in $3D$, one needs to create a more suitable representation of the environment. Specifically this representation should represent the full $3D$ environment including overhangs and concavities such as windows and doorways. In addition, the representation must be sufficiently dense that the planner does not mistakenly decide to fly through a wall.

While $2D$ laser scanners provide very rich information about the environment around the vehicle, they contain no information regarding the area above and below its sweep. However, since the vehicle is able to change height, we are able to plan actions that explore the space above and below the vehicle, such that we can obtain dense point clouds of

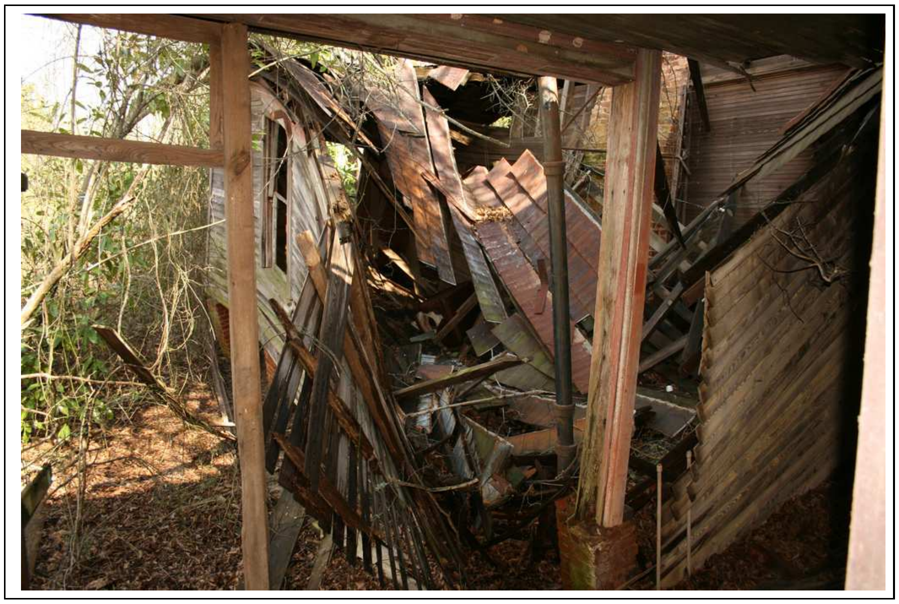


Figure 5-1: A room in a partially collapsed building. Flying in this room would likely be impossible with a $2D$ world model. [Photo credit: Sean Green. Picture taken of collapsing rural dwelling in Ora SC]

the surrounding environment, such as the one shown in figure 5-2. However, moving the vehicle is slow, and potentially dangerous since the mirrors deflecting beams above and below the vehicle will only detect obstacles immediately above and below the vehicle, and could easily miss an obstacle such as a low hanging light.

Fortunately, cameras capture data that contains rich information about the full $3D$ environment around the vehicle, although extracting this information is notoriously difficult, even for stereo imagery. This is particularly true in environments with large featureless areas such as blank walls, which can be common in many indoor environments. The existence of these types of environments eliminates purely feature based approaches, such as in common visual SLAM algorithms [72, 28]. In addition, the environments pose a problem for correlation based dense stereo [42]. So while, these methods have been used in very impressive work by Kagami et al [52] and Mordohai et al [65], using them on a MAV platform could prove dangerous. In our work, we combine the laser and camera sensors, so as to leverage each of their strengths.

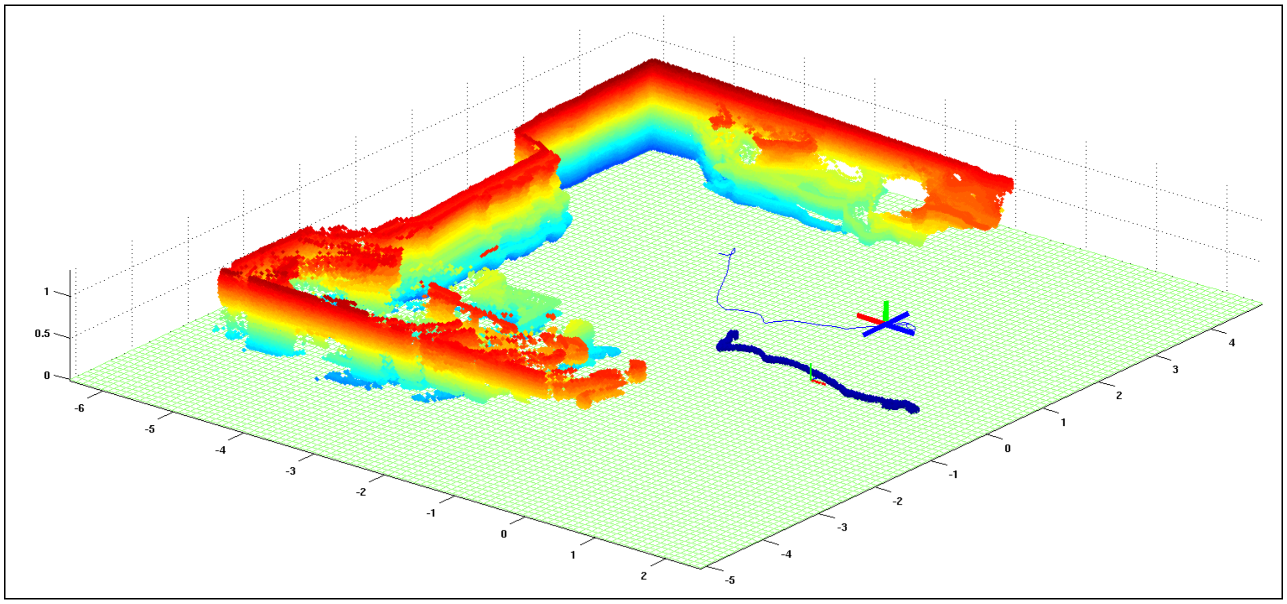


Figure 5-2: The $3D$ point cloud obtained by the vehicle performing a “height-sweep”.

5.1 Background

Over the years there has been considerable work on stereo algorithms. Most of this work has focused on either sparse feature based stereo such as was discussed in section 2.3, or dense correlation based approaches which seek to estimate the depth of every pixel in one image by matching a window around the pixel to a window in the second. However, more recently, researchers have used global image information to *infer* the depth of pixels for which there is no unambiguous correspondence between images [83, 30, 87]. This inference can framed as a Markov Random Field(MRF), which balances the the estimate of the individual pixel’s depth with the likelihood that it is at the same depth as similar neighboring pixels. This allows regions of low confidence to be filled in from neighboring regions with high confidence in a principled manner. Unfortunately these algorithms discretize the space of disparities such that the resolution obtained is fairly low. This effect is particularly problematic for horizontal planes such as the ground-plane, which would look like a staircase instead of a flat plane.

A more promising approach for our use is the work by Saxena et al [81] dubbed “Make3D”, which uses monocular cues to predict the $3D$ structure of a scene. While the major focus of their work is on learning a classifier model that is able to predict depth given only a single image, the classifier is used in conjunction with a MRF model that infers the overall scene structure from the classifier's noisy depth predictions. Their model segments the image into superpixels which are assumed to be planar faces of the environment. Although they make the assumption that the superpixels are planar, unlike the belief propagation based stereo algorithms mentioned above, they do not make the assumption that the planes are fronto-parallel. The MRF model then allows them to jointly reason about the most likely depth and orientation of each plane given the depth estimates predicted from monocular features as well as the relations between neighboring planes, such as coplanarity, connectedness, etc. While the output of their algorithm is visually pleasing, due to the inherent scale ambiguity of monocular image cues, by itself the algorithm would be challenging to use for planning. Fortunately as they showed in subsequent work, the MRF formulation allows them use depth information from other sensing modalities, such as stereo triangulation in addition to the monocular cues. They showed that the combination outperforms both individual approaches.

While Saxena et al's approach sought to predict the depth for a single image, to plan paths for the MAV, we seek to build a consistent $3D$ representation of the entire environment, including areas out of the the current field of view of the camera. In addition, we would like to incrementally build and improve the map as the MAV moves through the environment.

5.2 Our Approach

As an initial step towards solving this problem, we develop an approach that integrates both the laser and camera sensors so as to leverage on their respective strengths. The laser gives us very dense measurements in the 2D plane around the vehicle, while stereo gives sparse measurements of visually distinct features in $3D$. Finally monocular image cues give us information about the global structure of the environment.

We start by using monocular image cues to over-segment the environment into triangular patches which respect image intensity boundaries. By projecting the current set of laser and stereo points onto the image, we obtain measurements of the location and orientation for some of these planar patches. We then use a MRF to infer the most likely configuration of all triangular patches. Like Make3D, the MRF encourages neighboring patches to be connected and/or coplanar, while still respecting the measurements from the stereo and laser. Where our work differs from Make3D, is that our MRF models the full location of each surface patch, rather than the depth with respect to the camera. By estimating the full location without tying each triangle to a particular image, we can incrementally grow the triangular mesh over time as new images and range measurements arrive.

5.2.1 Triangular Segmentation

In order to efficiently model the environment around the vehicle, we need a representation that is compact, yet descriptive enough to model arbitrarily shaped surfaces. To this end, we chose a triangle mesh representation due to its simplicity and generality. A key issue in this selection is the fact that both the image space projection and the full $3D$ representation are simple and easy to reason about. When the current triangle mesh is projected into a subsequent image, it will result in a triangular segmentation of the regions of the image it covers.

Unfortunately image segmentation algorithms generally break an image into a set of amorphous blobs which attempt to capture distinct object boundaries. While these blobs can be used easily in image space, such as in Make3D, representing them in $3D$ is unwieldy. As a result, our first step was to develop a method for segmenting an image into a set of triangular regions that respect object boundaries to within some tolerance.

We first segment the image using the algorithm developed by Felzenszwalb et al [31], which gives us a segmentation such as the one shown in figure 5-3. The segmentation algorithm return the pixel boundaries between regions, which we join into a set of contours using a binary image edge-linking routine.

We then apply the Douglas-Peucker line simplification [29] algorithm to obtain a set of piecewise linear line-segments that approximate the segmentation boundaries as shown in figure 5-4. The algorithm preserves all junctions of 3 or more regions, and ensures that the approximate boundaries do not differ from the original by more than a given number

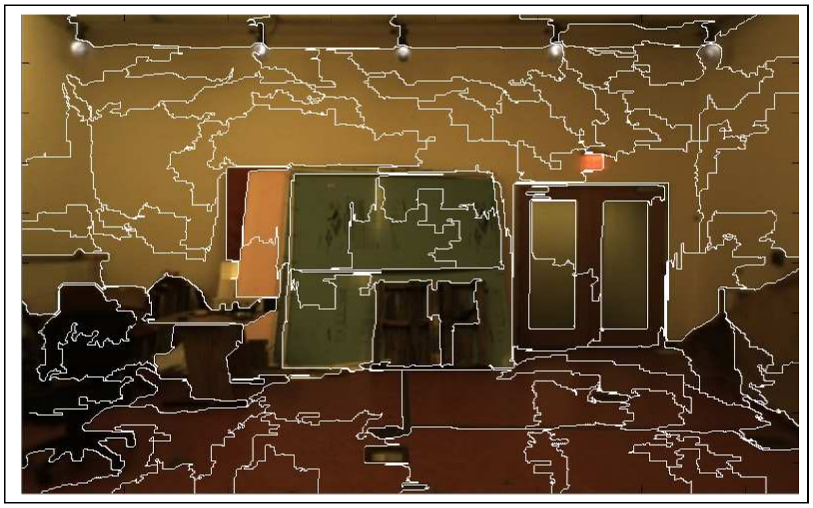


Figure 5-3: The original image segmented by [31].

of pixels. To remove small regions we merge endpoints that are within a given distance of each other.

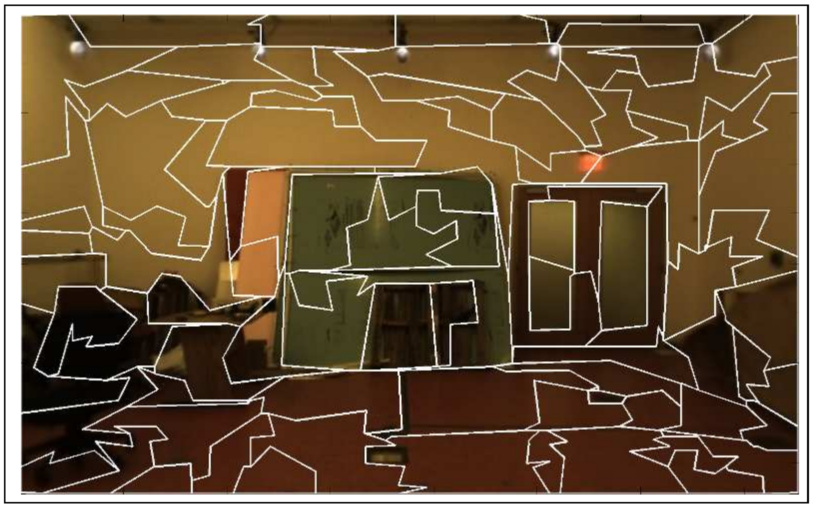


Figure 5-4: The Segmented image after applying the simplification routines.

Finally, we break these polygonal regions into triangles by performing a constrained Delauney triangulation using the open source Triangle package [85], resulting in the final triangular segmentation shown in figure 5-5.

For subsequent images after the first image is processed, the triangular segmentation is only applied in regions of the image which are not covered by the current triangular mesh. These regions are obtained by projecting the current mesh onto the new image to mark the covered regions. The uncovered regions will be composed of the borders of the image,

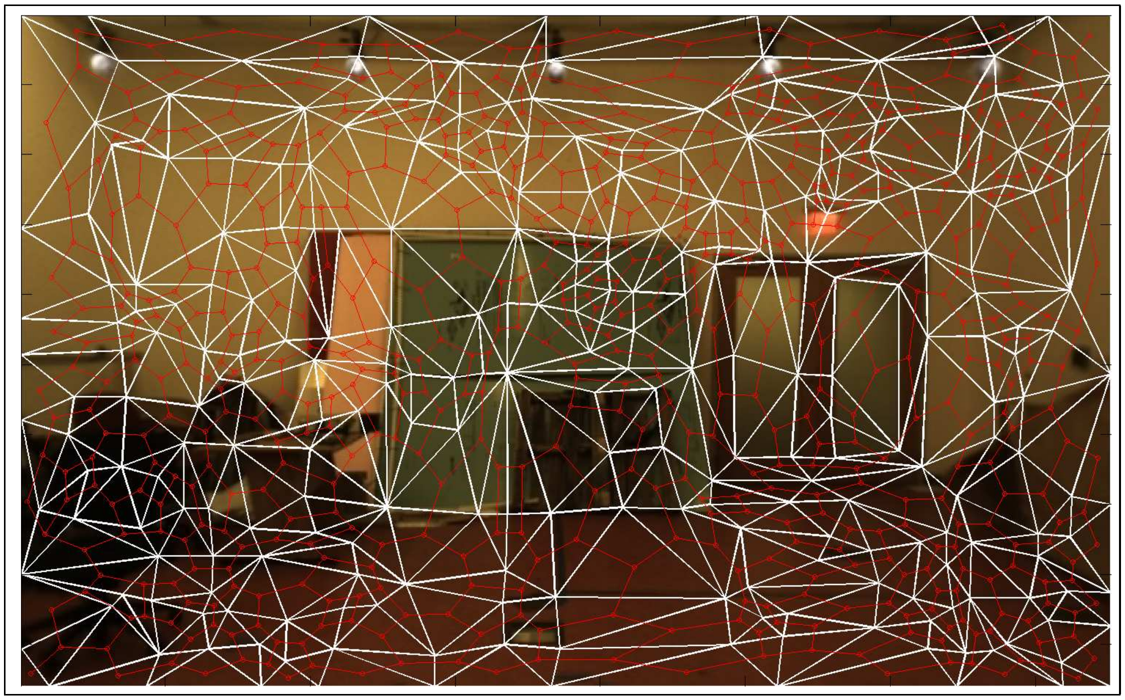


Figure 5-5: The final triangular segmentation of the image. The red lines are the nodes, and connectivity in the associated MRF.

locations where new surfaces enter the field of view due to camera motion, and regions that were previously occluded by a foreground object. The new triangles are assumed to be adjacent to their neighbor from the old mesh in image space. While this assumption could pose problems due to incorrect correspondences, we assume the vehicle is well localized, such that object boundaries in the new and old image match with reasonable accuracy.

5.2.2 Data Association

In addition to the monocular image information used for the segmentation, we have the range measurements obtained from the laser and stereo. To use these measurements, we assume that range measurements are points on a surface observed in the camera images. We associate each measurement with a specific triangular image patch by projecting the range measurement into the current camera image to identify which triangle it falls in. We then project the range measurement to the point in $3D$ using the current location of the vehicle.

This 3D point serves as a soft constraint on the location of its associated triangular patch in our MRF. An important feature of performing the data association in this manner is that it allows us to add measurements for planar patches after the image was captured. For example, to measure the vehicle height above the surface, we deflect some of the laser beams downwards. These beams hit surfaces that are out of the field of view of the camera. However, as the vehicle moves forward, they will hit surfaces that were in the field of view in previous camera images, providing very useful measurements for previously empty triangular patches. This effect is seen in the measurements at the bottom of the image in figure 5-6

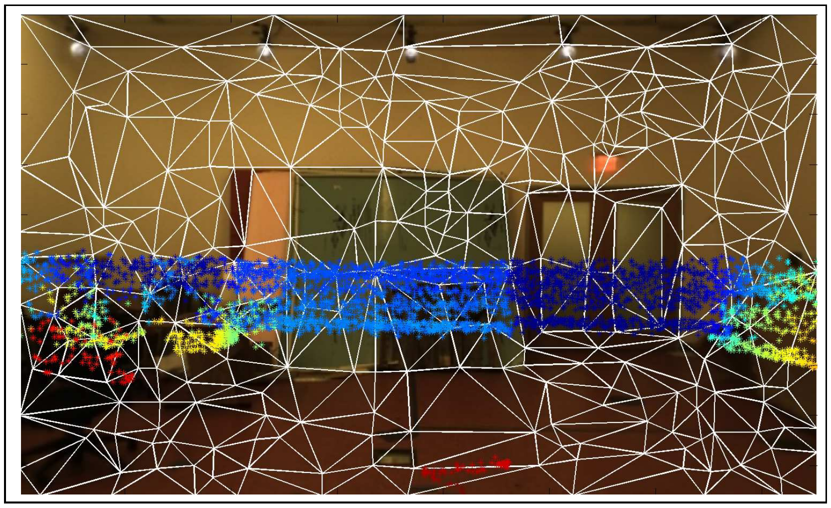


Figure 5-6: Laser points projected onto the camera image. The point color is a function of the distance from the camera (red is close).

5.2.3 Inference

Given a set of planar patches, some of which have associated point measurements, we wish to infer the most likely configuration of all patches. Since some patches do not have enough point measurements, the problem is under-constrained. However, not all configurations are equally likely. Adjacent patches are likely to be connected, except in the case of an occlusion boundary. Furthermore, adjacent patches are likely to be close to coplanar since we over-segment the images. This inference problem can be formulated as a Markov random field, with potential functions characterizing the likelihood of a given configuration of neighboring patches.

The MRF has three types of variables:

1. $z \in \Re^3$ are range measurement nodes
2. $e \in [0, 1]$ are discontinuity nodes
3. $y \in \Re^9$ are patch nodes

The first two types of nodes, z and e, are computed directly from the data, and are fixed given the observed data. The last type of node, y, are the variables that we are trying to estimate from the data. We parameterize the planar patch y by the $3D$ position of its three corners. The variables z represent known locations which should be on the surface represented by the associated planar patch y. Finally, the edge nodes e represent the likelihood of a discontinuity or fold occurring between two planar patches. We compute this likelihood as a function of the number of images in which the segmentation algorithm indicates that the two patches are part of the same segment. A normal configuration of these types of nodes is shown in figure 5-7.

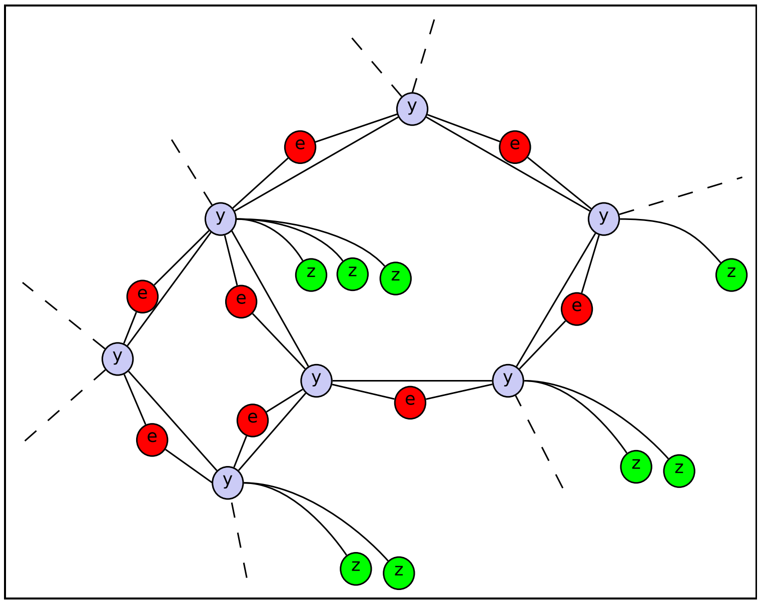


Figure 5-7: Part of the MRF used to model the environment.

Each clique of nodes has an associated potential function describing the likelihood of a surface configuration given by:

$$\Psi(y_i|Z_i) = exp(-\sum_{z_j \in Z_i} d(z_j, y_i)) \tag{5.1}$$

where Z_i is the set of measured ranges attached to path y_i, and $d(z_j, y_i)$ is the distance from point z_j to the planar patch y_i, and

$$\Phi(y_i, y_j|e_{i,j}) = exp(-e_{i,j} f(y_i, y_j)) \tag{5.2}$$

where $f(y_i, y_j)$ is a function of the angle between the normal vectors of the two planes, and the distance between the adjacent corners. The first potential encourages the model to fit the data well, while the second potential encourages "smooth" solutions. The resulting overall likelihood for the entire model is obtained taking the product of all clique potentials. The MRF is therefore defined as

$$P(y|z, e) = \frac{1}{\eta(z, e)} \prod_{i=1}^{N} \Psi(y_i|Z_i) \prod_{i,j=1}^{N} \Phi(y_i, y_j|, e_{i,j}) \tag{5.3}$$

where $\eta(z, E)$ is the normalization constant (partition function).

To initialize the model, we set all y_i for which $Z_i \neq \emptyset$ to be the least-squares fit of the measured data. For all other patches, we assume that the environment consists of vertical walls and a flat floor, and use the vehicle's current height, and laser scan to set their initial position and orientation.

While exact inference on belief graphs with cycles has been proven to be NP-complete, there are a number of approximate algorithms that have been shown to work well in practice. We hope that we will be able to use one of these approximation methods, such as loopy belief propagation. However, the continuous valued variables and non-linear potential functions make applying standard algorithms difficult. Identifying a method to perform inference in our model is left for future work.

Chapter 6

Object Tracking

While much of the work in this thesis focused on enabling a MAV to localize and control itself in indoor environments, this chapter describes an object tracking system that would allow the MAV to follow a person or other designated object as they move through the environment.

We wished to make our tracking system general, without relying on specific features such as color, shape, or motion to identify the object to be tracked. This precludes using general object detection algorithms, such as “people detectors” to initialize and track the target objects. Instead, we focused on object tracking, relying on a human operator to detect the initial appearance of each object in the scene, and then track the object in successive frames. While object tracking has been studied extensively in the computer vision community, attempting to track objects from a camera mounted on a MAV poses unique challenges that prevent us from using existing work directly. To perform object tracking from a MAV platform one must handle:

1. Fast camera motion from an unsteady MAV platform
2. Low resolution and low frame-rate cameras due to weight and bandwidth constraints
3. Small objects, since the MAV will usually be observing the object from a distance.

These challenges are in addition to the usual visual tracking challenges of changing appearance and lighting. These challenges eliminate many of the techniques for object tracking that are common in the literature. Low resolution and small objects size make performing descriptor based tracking [102] difficult since the objects to be tracked will generally not have enough pixels associated with them to generate a sufficient number of features to track.

Even more challenging than the low resolution is the fast camera motion (combined with low frame rates) which prevents the use of background subtraction methods [55], and also leads to failure when using common tracking algorithms such as mean-shift tracking [22], since they are not able to adapt fast enough to handle the camera motion. Fast camera motion also limits the amount of temporal smoothing that can be done to average multiple noisy estimates of the object location in the image, since the location in the image can change drastically between frames. A survey of tracking methods for other difficult situations is given in [73].

In order to meet the challenges of object tracking from a MAV platform, we developed a new object tracking system which explicitly handles the fast motion of the vehicle.We developed a modified version of the classifier-based adaptive ensemble tracker, developed by Avidan [11]. Our algorithm, which we call Agile Ensemble Tracking (AET), uses the same object appearance classifier; however, instead of using mean-shift to track the object across frames, we use a more robust, particle-filter based, Bayesian filter approach that it is able to handle the fast motion of the MAV-mounted camera. While our approach does not provide completely autonomous operation, it significantly reduces the amount of attention required from the operator for the MAV to track an object over time.

6.1 Learning Object Appearance Models

Once an initial estimate of the target object in an image is identified by a human operator, we use a machine learning classifier to learn a model of the object's appearance. The classifier is trained to distinguish pixels that belong to the object from background pixels. To train the classifier, we assume that the object is localized within a known $n \times m$ sub-block of the image; pixels within that sub-block are given positive labels, and pixels outside that sub-block are given negative labels. Each pixel is described by d local features, e.g., local color features and a histogram of local oriented gradient features [27]. Each pixel i at image location $\mathbf{p}_i$ is therefore a separate training instance consisting of a d-dimensional feature vector $\mathbf{x}_i \in X$ and a label $y_i \in Y$. To distinguish the object from the background, we learn a classifier that predicts the label for each pixel based on the local image features. Following Avidan's work, we use a boosting method inspired by AdaBoost [82] to learn this classifier. AdaBoost requires a weak classifier, which in this algorithm is implemented as a linear separating hyper-plane $\mathbf{h}$, such that

$$\hat{y}(\mathbf{x}_i) = h(\mathbf{x}_i) = \text{sign}(\mathbf{h}^T\mathbf{x}_i) \tag{6.1}$$

where $\hat{y}(\mathbf{x})$ is the classifier output label for instance $\mathbf{x}$. The separating hyper-plane for a set of examples is computed using weighted least squares. This weak classifier is then boosted to learn an ensemble of classifiers $H = \{\mathbf{h}_1, \ldots, \mathbf{h}_K\}$ with associated weights $\alpha_1, \ldots, \alpha_K$. K is the total number of classifiers that are maintained by the algorithm. These weights are chosen iteratively, as shown in Algorithm 7.

Algorithm 7 : ADABOOST

```
Require: $N$ training instances $\{\mathbf{x}_i, y_i\}$
1: Initialize weights $\{w_i\}_{i=1}^N$ to be $\frac{1}{N}$
2: for $k = 1 \ldots K$ do
3:     Normalize $\{w_i\}_{i=1}^N$ to sum to $1$
4:     Train weak classifier $\mathbf{h}_k$
5:     $err = \sum_{i=1}^N w_i |\mathbf{h}_k(\mathbf{x}_i) - y_i|$
6:     $\alpha_k = \frac{1}{2} log \frac{1-err}{err}$
7:     Update $w_i = w_i e^{\alpha_k |\mathbf{h}_k(\mathbf{x}_i) - y_i|}$ for $i = 1 \ldots n$
8: end for
9: return $H(\mathbf{x}_i) = \sum_{k=1}^{K} \alpha_k \mathbf{h}_k(\mathbf{x}_i)$
```

In order to capture the appearance characteristics of an object at different scales, we train a separate ensemble of classifiers for a range of image scales. We then classify the pixels of a new image using the multi-scale, boosted, ensemble classifier, such that each pixel receives a (normalized) weighted vote for each label from each classifier based on the local image features at each pixel. The output of the classifier is a new image where each pixel has an associated likelihood value that it belongs to the tracked object.

Figure 6-1(a) illustrates an example training image, where the pixels in the inner block are positive training instances and the pixels in the outer block are negative training instances. Figure 6-1(b) shows the weighted classifier response to the same image after training. Notice that pixels along the sharply distinct color boundaries have the greatest classifier response.

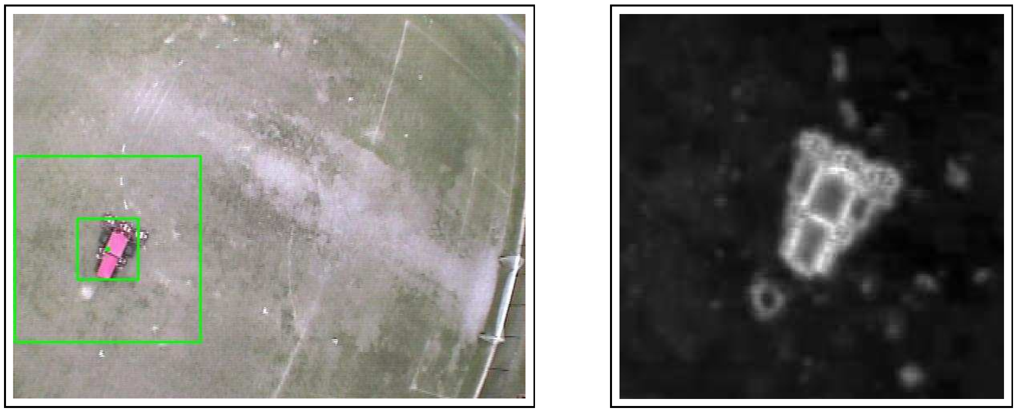

(a) Original Image (b) Ensemble Classifier Response

Figure 6-1: (a) An example training sub-block. The pixels in the smaller, inner block are assumed to be positive training instances, and the pixels in the outer block are negative training instances. (b) The response of the weighted classifiers across the sub-image of the detected vehicle. The intensity of each pixel is the likelihood of belonging to the object as provided by the classifier.

During tracking, the appearance of both the object and the background will vary over time; for instance, the orientation of edge features will change as objects rotate in the image frame. We therefore continually learn new classifiers from the incoming images. After tracking has been performed on each image, the image is used as a new training instance for learning new classifiers. Using boosting, the B best classifiers are retained from the current K classifiers, while $K - B$ additional classifiers are trained and added to the set of weak classifiers. This process of updating the classifiers is shown in Algorithm 8. In order to ensure that this retraining does not result in a drift over time away from the original image, we also investigated a variation where a subset of the original K classifiers are kept, regardless of the performance of this classifier subset at the current time step. This modification would ensure that there always exist at least some classifiers that are known to be correct.

Algorithm 8 : ADABOOST ONLINE UPDATE

```
Require: $N$ training instances $\{\mathbf{x}_i, y_i\}$, existing strong classifier $H_{in} = \{\mathbf{h}_1, \ldots, \mathbf{h}_K\}$
 1: Initialize weights $\{w_i\}_{i=1}^N$ to be $\frac{1}{N}$
 2: $H_{out} = \{\emptyset\}$
 3: for $k = 1 \ldots B$ do
 4:     Normalize $\{w_i\}_{i=1}^N$ to sum to $1$
 5:     for $\mathbf{h}_j \in H_{in}$ do
 6:         Compute $err_j = \sum_{i=1}^N w_i|\mathbf{h}_j(\mathbf{x}_i) - y_i|$
 7:     end for
 8:     Choose $\hat{\mathbf{h}} \in H_{in}$ with minimum $\widehat{err}$
 9:     $\hat{\alpha} = \frac{1}{2} log \frac{1-\widehat{err}}{\widehat{err}}$
10:     Remove $\hat{\mathbf{h}}$ from $H_{in}$ and add to $H_{out}$
11:     Update $w_i = w_i e^{\hat{\alpha}|\hat{\mathbf{h}}(\mathbf{x}_i) - y_i|}$ for $i = 1 \ldots n$
12: end for
13: for $k = B + 1 \ldots K$ do
14:     Normalize $\{w_i\}_{i=1}^N$ to sum to $1$
15:     Train weak classifier $h_k$ as in ADABOOST
16:     Add $h_k$ to $H_{out}$
17: end for
18: return $H_{out}$
```

6.2 Image Space Object Tracking

In the original ensemble tracker [11], the estimate of the object's location is found using mean-shift on the likelihood image computed from the classifier response. Starting from the previous target rectangle, mean-shift uses a hill-climbing technique to find the $m \times n$ rectangular region which contains the greatest aggregate response. While this approach works quite well for relatively stationary cameras, we found that the mean-shift approach was unable to handle the fast motion of our MAV platform.

As a result, we modified the tracking algorithm to use a particle-filter based Bayes filter to update the position estimate of the object. We incorporate an estimate of the camera ego-motion as a prior for predicting the location of the object in a subsequent image. This ego-motion estimate is essential for compensating for unpredictable motions of the camera, which would otherwise cause the tracker to lose track of the object. The attitude of the vehicle, as estimated by its onboard IMU, was too noisy to provide an adequate estimate of this ego-motion. Instead we estimate it directly from the imagery by computing optical flow between the entire previous and current images. We make use of the Pyramidal-Lucas-

Kanade optical flow implementation available in the OpenCV package [25]. The optical flow algorithm computes a sparse set of f feature matches $\{\mathbf{p}_i^{t-1}, \mathbf{p}_i^t\}_{i=1}^f$, where $\mathbf{p}_i^{t-1}$ is the $2D$ pixel location of feature i in the image I_{t-1}, and $\mathbf{p}_i^t$ is its corresponding location in image I_t at the next time-step. Using these feature matches, we can estimate the camera ego-motion as an a 3×2 affine transformation matrix Δ such that:

$$\mathbf{p}_i^{t+1} \approx \left[\begin{array}{cc} 1 & \mathbf{p}_i^t \end{array} \right] \Delta \tag{6.2}$$

This affine transformation captures translation, rotation, scaling, and shearing effects in image space. Due to the height of the vehicle, and the nearly planar ground surface, an affine transformation is generally a reasonable approximation.

Since some of the feature matches may be wrong or correspond to moving objects, we refine the ego-motion estimate by performing expectation-maximization (EM) to identify the affine transformation that best explains the apparent camera motion. Other methods such as RANSAC could also be used. The affine transformation Δ is then used in the motion model of a Bayes filter, while the learned object appearance model H is used in the associated sensor model. We use a particle filter to approximate the posterior distribution $p(\mathbf{p}_t|z_{0:t})$ according to

$$p(\mathbf{p}_t|z_{0:t}) = \alpha p(z_t|\mathbf{p}_t) \int_{X_{t-1}} p(\mathbf{p}_t|\mathbf{p}_{t-1}) p(\mathbf{p}_{t-1}|z_{0:t-1}) dt. \tag{6.3}$$

where $\mathbf{p}_t$ is the location of the object in the image at time t, z_t is the object measurement calculated from the image at time t, $p(\mathbf{p}_t|\mathbf{p}_{t-1})$ is our motion model, $p(z_t|\mathbf{p}_t)$ is our sensor model, and $p(\mathbf{p}_{t-1}|z_{0:t-1})$ is the prior distribution of the object's location.

The object measurement z_t is obtained by using the learned object appearance model to classify the image at time t. The classifier outputs a real value in the interval $[0, 1]$ for each pixel $\mathbf{q}_t$ in the image, and Figure 6-1(b) is a sample measurement. Our sensor model $p(z_t|\mathbf{p}_t)$ can therefore be characterized as follows,

$$z_t(\mathbf{q}_t)|\mathbf{p}_t = \begin{cases} 1 + \epsilon_z & \mathbf{q}_t = \mathbf{p}_t, \\ 0 + \epsilon_z & \text{otherwise,} \end{cases} \qquad \epsilon_z \sim N(0, \sigma_z), \tag{6.4}$$

where $z_t(\mathbf{q}_t)$ is the response of the classifier at pixel $\mathbf{q}_t$. Equation 6.4 essentially predicts that the classifier will respond with a 1 at the predicted location $\mathbf{p}_t$ in the image, and 0 everywhere else, where the measurements have Gaussian noise ϵ_z. The model is clearly approximate since the noise is not Gaussian (and measurements can never exceed 1), but the Gaussian model worked well experimentally.

It is computationally expensive to run the classifier on the entire image. Hence, we only run the classifier in the vicinity of the current particle filter mean estimate, and assume that the object has a minimal likelihood of being at all other locations in the image. Additionally, we smooth the classifier responses z_t across the image using a Gaussian blur operator to obtain a spatially smooth likelihood map, and each particle is given a weight equal to the value in the Gaussian-blurred probability image at its location in the image. Although this Gaussian smoothing creates minor correlations between image pixels, we continue to assume that the likelihood of object detection at each pixel is independent; experimentally the Gaussian smoothing of the classifier responses led to more robust object tracking even with this independence assumption, and more closely matched our Gaussian model of the classifier.

The motion model $p(\mathbf{p}_t|\mathbf{p}_{t-1})$ is equal to the ego-motion estimated from optical flow with additive Gaussian noise

$$\mathbf{p}_t|\mathbf{p}_{t-1} = \begin{bmatrix} 1 & \mathbf{p}_{t-1} \end{bmatrix} \Delta + \delta_{\mathbf{p}}, \qquad \delta_{\mathbf{p}} \sim N(0, \sigma_{\mathbf{p}}) \tag{6.5}$$

Algorithm 9 presents the complete Agile Ensemble Tracking algorithm. For clarity, although the algorithm is presented as if the images are all given to the algorithm at the start, on the real system, the images are actually processed in real-time as they are streamed from the vehicle.

In contrast to more conventional filtering techniques such as the Kalman filter [53], the particle filter is better at modeling the non-linearities in the sensor and motion models. In contrast to ground vehicles and fixed-wing aircraft that generally have stable attitudes, the attitude of the MAV is particularly dynamic and non-linear. Frequent attitude changes of the MAV would cause very large object displacements in the image.

Table 6.1(b) illustrates the benefits of the particle filter. Using the modified motion model, we were able to maintain a track of the person in Figure 6-2(b) for over 2 minutes, requiring human intervention only once when the person left the frame for a few seconds. In contrast, a much higher rate of human intervention to reacquire lost tracks was required when the original (non-optical-flow-based) motion prediction was used.

Algorithm 9 : AGILE ENSEMBLE TRACKING

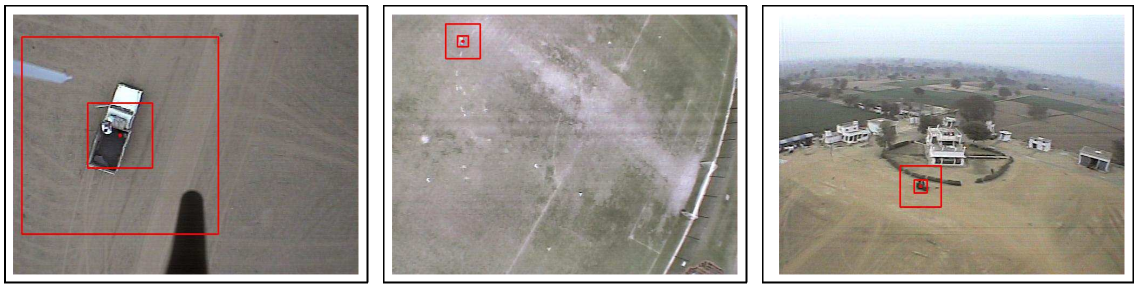
```
Require: $T$ video frames $I_1 \dots I_T$, initial object bounding box $r_1$
1: Learn initial strong classifier $H_1$ from $I_1$ and $r_1$ using ADABOOST
2: for $I_t = I_2 \dots I_T$ do
3:     Compute ego-motion estimate $\Delta$ from $I_{t-1}$ to $I_t$
4:     Propagate image space particle locations using $\Delta$
5:     Use $H_{t-1}$ to update the likelihood of each particle and perform importance sampling
6:     Use filter's maximum likelihood estimate as prediction of rectangle $r_t$
7:     Compute $H_t$ using ADABOOST ONLINE UPDATE
8: end for
```

6.3 Tracking Analysis

Human intervention is still required to ensure that the object is continuously being tracked, to potentially restart the tracker when it fails, and to initialize the tracker when new objects of interest appear. We evaluated the tracker under different configurations, including with and without the motion prediction given by optical flow, with and without retraining, as well as retaining different numbers of original classifiers. We tested the object tracker on very different targets across a wide variety of scenes, measuring the number of times that the estimate of the object's location diverged from hand-labeled, ground-truth data.

The easiest object tracking problem was the vehicle from overhead, shown in Figure 6-2(a). This data set contained 17 seconds of video, for a total of 250 frames.[1] Due to the large vehicle size, crisp features and stable hover of the MAV, we obtained good performance for all tracker configurations. As Table 6.1(a) reveals, even with the non-optical-flow motion model, or the online retraining of the classifier, the tracker never lost the vehicle

[1] We typically received data from the vehicle at 15 Hz, but this number varied depending on the characteristics of the local RF field.



(a) EOD Vehicle (b) Person (c) MAV ’08 Guard Vehicle

Figure 6-2: Examples of the variety of objects tracked. (a) A vehicle from overhead. (b) A walking person. (c) The guard vehicle from the MAV ’08 competition. (a) was relatively easy to track, but (b) and (c) required a better motion prediction model.

No optical flow, no retraining	0 Hz (0)
No retraining	0 Hz (0)
Keep first 3 classifiers	0 Hz (0)
Full retraining	0 Hz (0)

(a) 250 frames, 17 seconds

No opt. flow, no retrain.	0.140 Hz (21)
No retraining	0.040 Hz (6)
Keep first 3 classifiers	0.027 Hz (4)
Full retraining	0.007 Hz (1)

(b) 2683 frames, 150 seconds

No opt. flow, no retrain.	0.39 Hz (21)
No retraining	0.26 Hz (14)
Retain first 3 classifiers	0.28 Hz (15)
Full retraining	0.30 Hz (16)

(c) 1000 frames, 54 seconds

Table 6.1: Performance comparison for the Agile Ensemble Tracking algorithm, comparing the effect of retraining, and the optical flow based motion model. The frequency of required track re-initializations is shown, with the total number of errors in parentheses

after initialization. In addition, retaining different numbers of the original classifiers had no effect on the tracker’s performance for this target.

Tracking the walking person, shown in Figure 6-2(b), was much more challenging due to the small size of the person in the image. Nevertheless, by taking advantage of the ego-motion estimation, the AET algorithm was still able to achieve excellent performance. As Table 6.1(b) demonstrates, optical flow played an important role in keeping the tracking estimate on target. In addition, adapting the object appearance over time led to improved tracking. Although the appearance of the person moving around the field was relatively constant, the background changed dramatically when the person moved from the green grass to the gray dirt patches. Retraining and adapting the classifier therefore ensured that the classifier was able to maintain enough discrimination between the person and the background to continue tracking accurately.

Finally, we evaluated the tracker performance in tracking the guard vehicle in the MAV ’08 competition. With a forward-pointing camera, image changes between frames due to the MAV motion became more pronounced. In addition, as shown in Figure 6-2(c), the hedges surrounding building were exactly the same color and similar shape as the guard vehicle. As a result, the tracker lost track of the guard vehicle far more often than in the other data sets we tested on.

In this data set, the camera motion, rather than changes in appearance, was the major factor that resulted in the tracker becoming lost. The guard vehicle was moving slowly enough that its motion should have had a negligible effect. Instead, from watching the video of the guard vehicle, there were several situations where the pitching and rolling of the MAV caused abnormally large inter-frame motion. In some of these cases, the optical flow was able to estimate and compensate for this ego-motion. In others, however, the optical flow computation failed to compensate for the camera motion, and many of these large inter-frame motion coincided with the tracker losing track of the vehicle. As a result, retraining the classifiers actually reduced performance slightly, since newer classifiers in the ensemble were trained on bad data as the tracker began to get lost, thereby creating a positive feedback cycle from which the tracker could not recover. While it is clear that the optical flow plays an important role in keeping the tracking on target, the optical flow algorithm may be unable to capture the full camera motion in some domains, resulting in the classifier becoming lost.

Fundamentally, to solve the tracking problem in the face of potentially large inter-frame camera motion, more sophisticated object detection is needed. Once the ensemble-based tracker loses the target, there is no way to recover by using a local appearance-based tracker that is learned online, since any corruption of the current object estimate will be propagated forward. Subsequent classifiers would then get corrupted. As a result, an object detector with higher-level learned invariants is needed to recover from object tracker failures in the general case.

6.4 Person Following

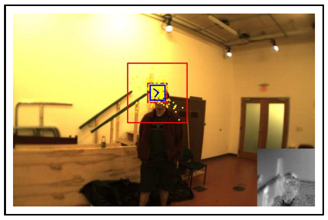
One possible use of the object tracker is to use it to provide high level commands, such as “follow this person”. To achieve this goal, we had to close the loop around controlling the vehicle using the object tracker. However, the object tracker described above performs its tracking purely in image space. To follow an object, we must first determine its position in the world, and then instruct the vehicle to move such that it keeps the object in the field of view.

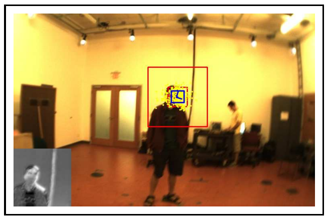
Given the tracked position of an object in the image, we can recover the position of the object in the world co-ordinates from knowledge of the intrinsic camera properties, such as camera focal length, center of projection, etc. if we know the depth. While the monocular camera used for the tracking does not give us this depth directly, we can estimate it using other sensors. We estimate the depth differently depending on whether we are indoors, with a forward facing camera as shown in figure 6-3 or outdoors, with the camera pointing down as shown in figure 6-2.

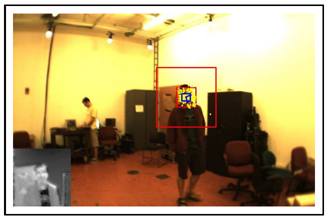
When the MAV is outdoors under GPS control, we use knowledge of the MAV GPS position and attitude along with an assumption that the ground plane is flat to estimate the depth. However, the GPS localization and attitude of the MAV are not known perfectly and in particular, small errors in attitude can lead to substantial errors in projecting from image co-ordinates to world co-ordinates.

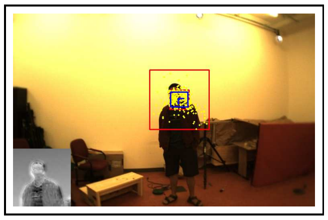
On the other hand, when indoors, with a forward facing camera, we can use the laser range-finder to estimate the depth. We know the relative transformation between the camera and laser, which allows us to use the camera to identify which laser beams are reflecting off of the object being followed. These laser beams can then be used to estimate the depth of the object being tracked.

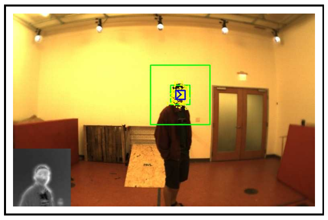
Since both of these methodologies provide noisy estimates of the objects location, we apply a second level of Bayesian filtering to maintain a cleaner estimate of the target location in the global coordinates. In contrast to the image-space filter, where we generally assume that the motion variance is large and emphasize the measurement model, when tracking in global coordinates, we place more weight on the motion model and model the projections from image coordinates to world coordinates as very noisy measurements. In this way, we average over many measurements to attain a more accurate estimate of the target location. Once we have this estimate of the target's global position, we can send a waypoint to the vehicle controller instructing it to move near the object.


(a) frame 1


(b) frame 28


(c) frame 57


(d) frame 97


(e) frame 141

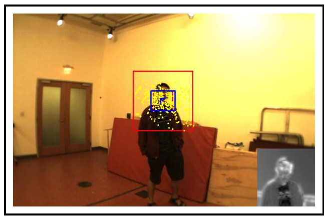

(f) frame 170

Figure 6-3: A sequence of frames showing the object tracking system closing the loop and controlling the vehicle to follow a person.

A simplified version of this closed loop following is shown in Figure 6-3. In this test, the object tracker controls the yaw of the vehicle, while hovering in one spot. The system manages to follow a person as they walk in a full circle around the vehicle.

Chapter 7

Conclusion

In this concluding chapter I give a brief summary of the work presented in this thesis before describing future work that would improve the capabilities of the system presented.

7.1 Summary

This thesis presents the system that we have developed to enable flight in unstructured and unknown indoor environments. As discussed in the introduction, this capability is very desirable, and would be of use in a number of scenarios such as disaster rescue, surveillance, and inspection among others. MAVs present a number of challenging problems that must be solved to enable indoor flight, however, as we have shown in the previous chapters, by carefully examining the algorithmic requirements for the system, these challenges can be overcome.

Our solution leverages the different real-time requirements for controlling the vehicle at multiple layers to develop a working system. At the base level, we use a very capable hardware platform that has onboard attitude stabilization. From there we developed the set of relative position estimation algorithms described in chapter 2 which provide high quality real-time motion estimates. These motion estimates were then fused with measurements from the onboard IMU to provide high quality real-time estimates of the vehicles position and velocity that were sufficient for local control and stabilization of the vehicle. With this base, we could integrate a SLAM module that closed loops and provided globally consistent state estimates enabling flight in large scale environments. In addition, the SLAM module provides a $2D$ map of the environment that enables autonomous planning and exploration behaviors.

While the $2D$ world model used in the work described above proved sufficient for a range of uses, to enable planning actions in the full $3D$ environment we need a more complete environment representation. Towards this end, we started developing a framework for performing dense reconstruction of the full $3D$ environment around the vehicle. This work is still in progress, however we believe that the framework outlined in chapter 5 will prove useful with more time.

Finally as a demonstration of high level closed loop autonomy, we developed the object tracking system described in chapter 6 which allows the vehicle to autonomously follow a person.

7.2 Future Work

While the system that we have developed is very capable, indoor flight is by no means a "solved" problem. As it stands right now, the system provides a MAV platform that would enable a human operator to easily fly the vehicle through a first-person-view onboard camera. However, there is still a tremendous amount of work to be done, to improve the robustness and autonomy capabilities of indoor MAVs.

1. **3D Capabilities:** Perhaps the most important direction of future work is extending the vehicle's capabilities in perceiving, planning, and operating in $3D$ environments. Replacing the $2D$ SLAM implementation with a fully $3D$ visual SLAM solution would improve the flight capabilities considerably, and make the state estimates globally consistent in $3D$. In addition, completing the framework developed in chapter 5 to provide a dense $3D$ environment representation will enable planning actions in $3D$.

2. **Faster Speed:** One of the appeals of MAVs is that they are capable of high speeds relative to their ground vehicle counterparts. However, the system as it is currently de-

signed assumes that the vehicle always remains within the hover regime, and moves slowly through the environment. While the quadrotor vehicles are capable of higher speeds, the sensing and control algorithms currently employed are not. More work will need to be done to be able to maintain high quality state estimation in the face of motion blur and reduced reaction time. Increasing the speeds will also likely require improving the control model and controller employed by our system beyond the simple LQR-based feedback controller.

3. **Onboard Computation:** The current system setup performs most of the computation offboard at the ground-station. This is a major limitation of the system. Using offboard computation limits the effective range of the vehicle since the bandwidth required to send the sensor data to the ground-station is more than long-range commercially available wireless links currently provide. Moving all computation onboard would improve the system performance and make the system far more robust. While computer hardware will continue to get faster, more work should be done to explore what modifications to the algorithms are possible that would improve their computational efficiency.

Bibliography

[1] Ascending technologies GmbH. http://www.asctec.de.

[2] Carmen - Robot Navigation Toolkit. http://carmen.sourceforge.net/.

[3] KFilter - free C++ extended Kalman filter library. http://kalman.sourceforge.net/.

[4] LCM - lightweight communications and marshalling. http://lcm.googlecode.com/.

[5] P. Abbeel, A. Coates, M. Montemerlo, A.Y. Ng, and S. Thrun. Discriminative training of Kalman filters. In *Proc. RSS*, 2005.

[6] Markus Achtelik. Vision-based pose estimation for autonomous micro aerial vehicles in gps-denied areas. Master's thesis, Technical University, Munich, May 2009.

[7] S. Ahrens. Vision-based guidance and control of a hovering vehicle in unknown environments. Master's thesis, MIT, June 2008.

[8] E. Altug, J. P. Ostrowski, and R. Mahony. Control of a quadrotor helicopter using visual feedback. In *Proceedings IEEE International Conference on Robotics and Automation*, volume 1, pages 72–77 vol.1, 2002.

[9] E. Altug and C. Taylor. Vision-based pose estimation and control of a model helicopter. In *Proceedings of the IEEE International Conference on Mechatronics*, pages 316–321, 2004.

[10] G. Angeletti, J.R. P. Valente, L. Iocchi, and D. Nardi. Autonomous indoor hovering with a quadrotor. In *Workshop Proc. SIMPAR*, pages 472–481, Venice, Italy, 2008.

[11] Shai Avidan. Ensemble tracking. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[12] A. Bachrach, A. Garamifard, D. Gurdan, R. He, S. Prentice, J. Stumpf, and N. Roy. Co-ordinated tracking and planning using air and ground vehicles. In *Proc. ISER*, 2008.

[13] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Proceedings of European Conference on Computer Vision*, pages 404–417, 2006.

[14] S. Bouabdallah, P. Murrieri, and R. Siegwart. Towards autonomous indoor micro VTOL. Autonomous Robots, Vol. 18, No. 2, March 2005.

[15] Jean-Yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm. Technical report, Intel Corporation Microprocessor Research Labs, 2000.

[16] J. E. Bresenham. Algorithm for computer control of a digital plotter. *Seminal graphics: poineering efforts that shaped the field*, pages 1–6, 1998.

[17] D.W. Casbeer, R.W. Beard, T.W. McLain, Sai-Ming Li, and R.K. Mehra. Forest fire monitoring with multiple small uavs. In *Proceedings of the American Control Conference*, pages 3530–3535 vol. 5, June 2005.

[18] M. Castillo-Effen, C. Castillo, W. Moreno, and K. P. Valavanis. Control fundamentals of small / miniature helicopters - A survey. In *Advances in Unmanned Aerial Vehicles*, volume 33, pages 73–118. Springer Netherlands, 2007.

[19] K. Celik, Soon J. Chung, and A. Somani. Mono-vision corner slam for indoor navigation. In *IEEE International Conference on Electro/Information Technology*, pages 343–348, 2008.

[20] Adam Coates, Pieter Abbeel, and Andrew Y. Ng. Learning for control from multiple demonstrations. In *Proceedings of the 25th International Conference on Machine Learning*, pages 144–151, New York, NY, USA, 2008. ACM.

[21] D. M. Cole and P. M. Newman. Using laser range data for 3D SLAM in outdoor environments. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1556–1563, 2006.

[22] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 142–149 vol.2, 2000.

[23] Intel Corporation. Intel integrated performance primitives. `http://software.intel.com/en-us/intel-ipp`.

[24] Intel Corporation. Intel math kernel library. `http://software.intel.com/en-us/intel-mkl`.

[25] Intel Corporation. Open source computer vision library. http://www.intel.com/technology/computing/opencv/index.htm.

[26] M. Cummins and P. Newman. Accelerated appearance-only SLAM. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1828–1833, 2008.

[27] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[28] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera SLAM. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.

[29] D.H. Douglas and T.K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10:112–122, 1973.

[30] Felzenszwalb, Pedro, Huttenlocher, and Daniel. Efficient belief propagation for early vision. *International Journal of Computer Vision*, 70(1):41–54, October 2006.

[31] P.F. Felzenszwalb and D.P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181, 2004.

[32] T. Furukawa, F. Bourgault, B. Lavis, and H.F. Durrant-Whyte. Recursive Bayesian search-and-tracking using coordinated UAVs for lost targets. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2521–2526, May 2006.

[33] LiPPERT Embedded Computers GmbH. `http://www.lippertembedded.com`.

[34] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[35] S. Grzonka, G. Grisetti, and W. Burgard. Autonomous indoor navigation using a small-size quadrotor. In *Workshop Proc. SIMPAR*, pages 455–463, Venice, Italy, 2008.

[36] S. Grzonka, G. Grisetti, and W. Burgard. Towards a navigation system for autonomous indoor flying. In *IEEE International Conference on Robotics and Automation*, pages 2878–2883, May 2009.

[37] Gumstix. Gumstix verdex. http://www.gumstix.com.

[38] D. Gurdan, J. Stumpf, M. Achtelik, K.-M. Doth, G. Hirzinger, and D. Rus. Energy-efficient autonomous four-rotor flying robot controlled at 1 kHz. *Proc. ICRA*, pages 361–366, April 2007.

[39] Dirk Haehnel. *Mapping with Mobile Robots*. PhD thesis, 2004.

[40] Chris Harris and Mike Stephens. A combined corner and edge detector. In *The Fourth Alvey Vision Conference*, pages 147–151, 1988.

[41] R. He, S. Prentice, and N. Roy. Planning in information space for a quadrotor helicopter in a GPS-denied environments. In *Proc. ICRA*, pages 1814–1820, Los Angeles, CA, 2008.

[42] Heiko Hirschmüller. Improvements in real-time correlation-based stereo vision, 2001.

[43] G.M. Hoffmann, H. Huang, S.L. Waslander, and C.J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proc. of GNC*, Hilton Head, SC, August 2007.

[44] Hokuyo. Hokuyo URG-04LX scanning rangefinder. https://www.hokuyo-aut.jp/.

[45] J. How, B. Bethke, A. Frank, D. Dale, and J. Vian. Real-time indoor autonomous vehicle test environment. *Control Systems Magazine, IEEE*, 28(2):51–64, 2008.

[46] A. Howard. Real-time stereo visual odometry for autonomous ground vehicles. In *Proc. IROS*, 2008.

[47] Stefan Hrabar and Gaurav Sukhatme. Vision-based navigation through urban canyons. *J. Field Robot.*, 26(5):431–452, 2009.

[48] Albert S. Huang, Edwin Olson, and David Moore. Lightweight communications and marshalling for low-latency interprocess communication. Technical Report MIT-CSAIL-TR-2009-041, Massachusetts Institute of Technology, 2009.

[49] NG. Johnson. Vision-assisted control of a hovering air vehicle in an indoor setting. Master's thesis, BYU, 2008.

[50] S.J. Julier, J.K. Uhlmann, and H.F. Durrant-Whyte. A new approach for filtering nonlinear systems. In *Proceedings of the American Control Conference*, volume 3, pages 1628–1632 vol.3, Jun 1995.

[51] Myungsoo Jun, S. I. Roumeliotis, and G. S. Sukhatme. State estimation of an autonomous helicopter using Kalman filtering. In *Intelligent Robots and Systems, 1999. IROS '99. Proceedings. 1999 IEEE/RSJ International Conference on*, volume 3, pages 1346–1353 vol.3, 1999.

[52] S. Kagami, Y. Takaoka, Y. Kida, K. Nishiwaki, and T. Kanade. Online dense local 3D world reconstruction from stereo image sequences. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3858–3863, 2005.

[53] Emil Kalman, Rudolph. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[54] C. Kemp. *Visual Control of a Miniature Quad-Rotor Helicopter*. PhD thesis, Churchill College, University of Cambridge, 2006.

[55] Zuwhan Kim. Real time object tracking based on dynamic feature grouping with background subtraction. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, 2008.

[56] Georg Klein and David Murray. Improving the agility of keyframe-based SLAM. In *ECCV '08: Proceedings of the 10th European Conference on Computer Vision*, pages 802–815, Berlin, Heidelberg, 2008. Springer-Verlag.

[57] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams. A perception driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10), September 2008.

[58] Kai Lingemann, Andreas Nchter, Joachim Hertzberg, and Hartmut Surmann. High-speed laser localization for mobile robots. *Robotics and Autonomous Systems*, 51(4):275–296, 2005.

[59] M.I.A. Lourakis and A.A. Argyros. The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm. Technical Report 340, Institute of Computer Science - FORTH, Heraklion, Crete, Greece, Aug. 2004.

[60] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[61] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry. *An Invitation to 3D Vision: From Images to Geometric Models*. Springer Verlag, 2003.

[62] A. Matsue, W. Hirosue, H. Tokutake, S. Sunada, and A. Ohkura. Navigation of small and lightweight helicopter. *Japan Society of Aeronautical Space Sciences Transactions*, 48:177–179, 2005.

[63] Masayoshi Matsuoka, Alan Chen, Surya P. N. Singh, Adam Coates, Andrew Y. Ng, and Sebastian Thrun. Autonomous helicopter tracking and localization using a self-surveying camera array. *Int. J. Rob. Res.*, 26(2):205–215, 2007.

[64] Luis O. Mejias, Srikanth Saripalli, Pascual Cervera, and Gaurav S. Sukhatme. Visual servoing for tracking features in urban areas using an autonomous helicopter. In *IEEE International Conference on Robotics and Automation*, pages 2503–2508, 2006.

[65] P. Mordohai, JM. Frahm, A. Akbarzadeh, C. Engels, D. Gallup, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewnius, H. Towles, G. Welch, R. Yang, M. Pollefeys, and D. Nistr. Real-time video-based reconstruction of urban environments. 2007.

[66] L. P. Morency and T. Darrell. Stereo tracking using ICP and normal flow constraint. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 4, pages 367–372 vol.4, 2002.

[67] Christoph Neumann and Vincent Roca. Analysis of FEC codes for partially reliable media broadcasting schemes. pages 108–119. 2004.

[68] Andreas Nüchter, Hartmut Surmann, Kai Lingemann, Joachim Hertzberg, and S. Thrun. 6D SLAM with an application in autonomous mine mapping. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1998–2003, 2004.

[69] E.B. Olson. Real-time correlative scan matching. In *IEEE International Conference on Robotics and Automation*, pages 4387–4393, May 2009.

[70] Edwin Olson. *Robust and Efficient Robotic Mapping*. PhD thesis, MIT, Cambridge, MA, USA, June 2008.

[71] OpenSlam. http://openslam.org/.

[72] L. M. Paz, P. Pinies, J. D. Tardos, and J. Neira. Large-scale 6-DOF slam with stereo-in-hand. *Robotics, IEEE Transactions on*, 24(5):946–957, 2008.

[73] Fatih Porikli. Achieving real-time object detection and tracking under extreme conditions. *Journal of Real-Time Image Processing*, 1(1):33–40, October 2006.

[74] Ryan M. Rifkin and Ross A. Lippert. Notes on regularized least-squares. Technical Report MIT-CSAIL-TR-2007-025, MIT, 2007.

[75] J.F. Roberts, T. Stirling, J.C. Zufferey, and D. Floreano. Quadrotor using minimal sensing for autonomous indoor flight. In *Proc. EMAV*, 2007.

[76] Edward Rosten and Tom Drummond. Machine learning for high speed corner detection. In *9th European Conference on Computer Vision*, May 2006.

[77] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *Proceedings of the Third Intl. Conf. on 3D Digital Imaging and Modeling*, pages 145–152, 2001.

[78] R. San Jose Estepar, A. Brun, and C.-F. Westin. Robust generalized total least squares iterative closest point registration. In *Seventh International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'04)*, Lecture Notes in Computer Science, pages 234–241, Rennes - Saint Malo, France, September 2004.

[79] S. Saripalli, J. M. Roberts, P. I. Corke, G. Buskey, and G. S. Sukhatme. A tale of two helicopters. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 805–810 vol.1, 2003.

[80] Srikanth Saripalli and Gaurav S. Sukhatme. Landing a helicopter on a moving target. In *IEEE International Conference on Robotics and Automation*, pages 2030–2035, 2007.

[81] A. Saxena, Min Sun, and A. Y. Ng. Make3d: Learning 3D scene structure from a single still image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):824–840, 2009.

[82] R.E. Schapire. The boosting approach to machine learning: An overview. *Lecture Notes in Statistics*, pages 149–172, 2003.

[83] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47(1-3):7–42, 2002.

[84] S. Scherer, S. Singh, L. Chamberlain, and S. Saripalli. Flying Fast and Low Among Obstacles. In *Proc. ICRA*, pages 2023–2029, 2007.

[85] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[86] B. Steder, G. Grisetti, S. Grzonka, C. Stachniss, A. Rottmann, and W. Burgard. Learning maps in 3D using attitude and noisy vision sensors. In *Proc. IROS*, pages 644–649, 2007.

[87] Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum. Stereo matching using belief propagation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(7):787–800, 2003.

[88] Imaging Development Systems. uEye LE Cameras. http://www.ids-imaging.com/.

[89] H. Takeda and J.C. Latombe. Sensory uncertainty field for mobile robot navigation. *Proc. ICRA*, 1992.

[90] T. Templeton, D.H. Shim, C. Geyer, and S.S. Sastry. Autonomous Vision-based Landing and Terrain Mapping Using an MPC-controlled Unmanned Rotorcraft. In *Proc. ICRA*, pages 1349–1356, 2007.

[91] S. Thrun. Learning occupancy grids with forward models. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS'2001)*, Hawaii, 2001.

[92] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.

[93] J. Tisdale, A. Ryan, Z. Kim, D. Tornqvist, and J.K. Hedrick. A multiple UAV system for vision-based search and localization, 2008.

[94] P. H. S. Torr and A. Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78:2000, 2000.

[95] G.P. Tournier, M. Valenti, J.P. How, and E. Feron. Estimation and control of a quadrotor vehicle using monocular vision and Moirè patterns. In *Proc. of AIAA GNC, Keystone, Colorado*, 2006.

[96] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *ICCV '99: Proceedings of the International Workshop on Vision Algorithms*, pages 298–372, London, UK, 2000. Springer-Verlag.

[97] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 13(4):376–380, Apr 1991.

[98] Vicon. Vicon motion capture systems. http://www.vicon.com/.

[99] G. Xylomenos, G. C. Polyzos, P. Mahonen, and M. Saaranen. TCP performance issues over wireless links. *Communications Magazine, IEEE*, 39(4):52–58, 2001.

[100] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proc. CIRA*, pages 146–151. IEEE Computer Society Washington, DC, USA, 1997.

[101] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152, 1994.

[102] H. Zhou, Y. Yuan, and C. Shi. Object tracking using SIFT features and mean shift. *Computer Vision and Image Understanding*, August 2008.