Vision-Based Guidance and Control of a Hovering Vehicle in Unknown Environments

by

Spencer Greg Ahrens

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
[June 2008]
May 2008

Author .....
Department of Mechanical Engineering
May 23, 2008

Certified by .....
Jonathan P. How
Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
John J. Leonard
Professor of Mechanical and Ocean Engineering
Faculty Reader

Accepted by .....
Lallit Anand
Chairman, Department Committee on Graduate Students

Vision-Based Guidance and Control of a Hovering Vehicle in Unknown Environments

by

Spencer Greg Ahrens

Submitted to the Department of Mechanical Engineering
on May 23, 2008, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

This thesis presents a methodology, architecture, hardware implementation, and results of a system capable of controlling and guiding a hovering vehicle in unknown environments, emphasizing cluttered indoor spaces. Six-axis inertial data and a low-resolution onboard camera yield sufficient information for image processing, Kalman filtering, and novel mapping algorithms to generate a high-performance estimate of vehicle motion, as well as an accurate three-dimensional map of the environment. This combination of mapping and localization enables a quadrotor vehicle to autonomously navigate cluttered, unknown environments safely. Communication limitations are considered, and a hybrid control architecture is presented to demonstrate the feasibility of combining separated proactive offboard and reactive onboard planners simultaneously, including a detailed presentation of a novel reactive obstacle avoidance algorithm and preliminary results integrating the MIT Darpa Urban Challenge planner for high-level control. The RAVEN testbed is successfully employed as a prototyping facility for rapid development of these algorithms using emulated inertial data and offboard processing as a precursor to embedded development. An analysis of computational demand and a comparison of the emulated inertial system to an embedded sensor package demonstrates the feasibility of porting the onboard algorithms to an embedded autopilot. Finally, flight results using only the single camera and emulated inertial data for closed-loop trajectory following, environment mapping, and obstacle avoidance are presented and discussed.

Thesis Supervisor: Jonathan P. How
Title: Professor of Aeronautics and Astronautics

Faculty Reader: John J. Leonard
Title: Professor of Mechanical and Ocean Engineering

Acknowledgments

I would like to thank the many people who have made this thesis possible. My family and friends, an endless source of encouragement, support, and livelihood; and the Aerospace Controls Lab, a research group of exceptional motivation, ability, and encouragement. Specifically, I would like to thank my advisor, Professor Jonathan How, for his guidance over the past 17 months and his assistance writing this thesis. I am grateful for Ray He, an extraordinary colleague who has provided extensive assistance in extending the hardware platform used to generate the results in this thesis, and Gregory Andrews, an exemplary engineer who has worked tirelessly in the development of the navigation filter. Finally, I would like to thank Eva Markiewicz for her deepest love and compassion. My life is infinitely richer and more joyous because of her.

I would also like to acknowledge Draper Laboratories, which has made my education and this thesis possible through the University Research and Development program. My education has also been sponsored in part by Boeing North America, Incorporated under contract title "Vehicle And Vehicle System Health Management." Finally, AFOSR supported this research in part with DURIP grant FA9550-07-1-0321.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Advancements in sensor technology and embedded electronics continually increase the capabilities of autonomous vehicles, decreasing vehicle size, increasing maneuverability, and broadening the range of operable environments. Figures 1-1 and 1-2 show

Figure 1-1: Commercial micro helicopter toy [19].

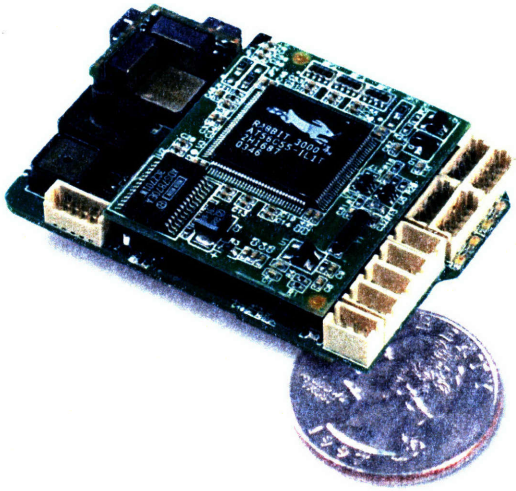


Figure 1-2: 16.7 gram fully-featured autopilot [53].

how this miniaturization is readily accessible in Commercial, Off-The-Shelf (COTS) hardware. Most of these advancements can be applied to many different vehicle types, all with unique advantages, limitations, and challenges. Unmanned Ground Vehicles (UGVs), for example, have demonstrated incredibly complex and robust mapping and navigation capabilities in a wide variety of indoor and outdoor environments,

but are limited in speed and maneuverability. Unmanned Air Vehicles (UAVs) have demonstrated complex autonomy and aggressive flight outdoors, as well as obstacle avoidance and simple target tracking on Micro Air Vehicles (MAVs) indoors, but are limited in payload.

Recently, a significant body of research has formed aiming to combine all of these capabilities to develop vehicle systems capable of complex autonomy and high maneuverability in indoor environments, spawning several conferences, symposiums, competitions, and summits to provide a forum for the large volume of work from academic, government, and private sectors [12, 17, 41, 42, 50]. One of the primary difficulties quickly discovered in this area of research is the complexity of developing new hardware, and the implementation of the necessary algorithms on embedded platforms. The computational limitations of these embedded systems force the development to focus on code optimization for speed, rather than on the performance of the core algorithms [18]. These limitations also restrict real-time feedback to the developer, making it difficult to troubleshoot problems during development.

The goal of this thesis is to describe the methodology and design of a system for a highly-capable autonomous vehicle, and present implementation details and physical results of many of the associated technologies and algorithms. We show the utility of the RAVEN testbed for leapfrogging many of the challenges that have traditionally slowed development in this field, allowing us to approach the core challenges more directly, advancing the solutions for later integration with embedded avionics. This will form a framework and path toward a complete system of incredibly high capability, and demonstrates a few breakthrough capabilities in the interim. Although the implementation is not yet complete, the architecture is planned such that full capabilities can be integrated efficiently in the future.

1.1 System Capabilities and Requirements

The term high-capability is best defined with an example scenario. A typical mission begins with an outdoor launch in an urban canyon. After a quick flight over the city,

the vehicle drops down to an upper story window of a collapsing office building, flies in through a broken window, and navigates the chaotic interior, beaming video and a real-time 3D map down to a ground station. After investigating the building, the vehicle flies back out the window and returns to base.

This scenario requires several technologies to be working in harmony, on both the algorithmic and physical levels. The first is the vehicle—it must be able to fly in order to cruise quickly over the city and reach a top-story window. It also needs to be small and highly maneuverable in order to navigate precisely inside the constrained, cluttered office space, such as the fire damaged NOAA building in Figure 1-3. We have identified the required vehicle to be a Hover-capable Micro Air Vehicle (HMAV). Although it is not explicit in the example above that the vehicle be capable of hovering, the ability to hover is desirable because of the increased utility of video from a semi-stationary platform, and omnidirectional acceleration helps in avoiding obstacles and navigating challenging spaces.

Figure 1-3: Cluttered, constrained, and dangerous space after a fire at NOAA's National Severe Storms Laboratory [51].

Secondly, the environment inside the building is completely unknown and must be mapped in three-dimensions for the vehicle to effectively plan its movement. To accomplish this, the vehicle must carry a sensor package capable of accurately perceiving the environment, but that is sufficiently small, light, and low-power so as to be carried a long distance. This sensor package must also provide enough information such that the motion of the vehicle can be estimated with sufficient accuracy to control the vehicle, stabilizing the fast, nonlinear dynamics, and accurately tracking trajectories.

Finally, the vehicle must be able to transmit data and accept commands from a remote operator, but also be robust to inevitable communication loss. This means

there must be a wireless link between the vehicle and the ground station, but the vehicle must be capable of self-contained autonomy when the communication link fails. Although advanced, computationally intensive path planning can be done on the ground station and transmitted to the vehicle, the vehicle must be able to reacquire this connection with autonomy that can function on the limited embedded hardware onboard the vehicle.

1.2 Motivation

The rewards of such capabilities are extensive, and would enable many missions so far impossible with current technology. Disaster areas such as those created in the wake of hurricane Katrina could be quickly searched from high altitude, after which the same vehicles could fly down and into flooded or partially collapsed buildings, looking for survivors, severed gas lines, and other time sensitive situations. Damaged structures such as the office building in Figure 1-4 are often unsafe for rescue-workers to enter, thus the unmanned vehicles could provide vital, time-sensitive information without risking additional lives. These autonomous vehicles could also investigate difficult to reach or hazardous places, such as the undersides of bridges or contaminated service tunnels, inspecting them for damage and tampering. These vehicle could also cover vast, complex areas quickly and effectively, dropping down between buildings in urban canyons, and among trees under forest canopies in pursuit of criminals or lost persons.

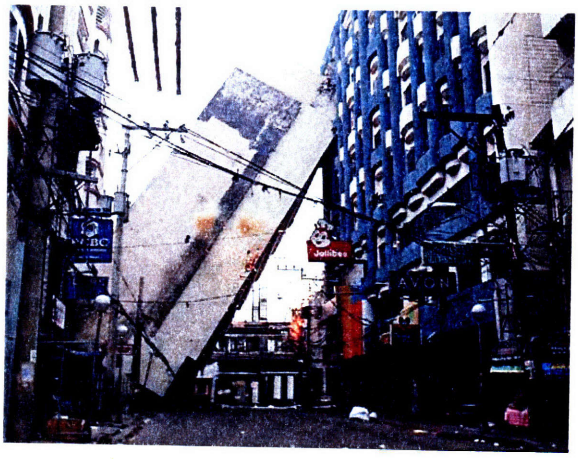


Figure 1-4: A partially collapsed building in Manila, Philippines that would be unsafe for human rescue-workers [9].

Because of the general formulation of the problem and the strict performance requirements necessary to navigate in three-dimensions on a dynamically unstable vehicle, most all of the algorithms and associated technologies developed here can

by applied to many other mobile robotic systems. One example is UGVs, which are typically large and bulky to accommodate dense sensor suites of LIDAR, RADAR, cameras, and more, but the system described here would enable comparable mapping and autonomy capabilities in a much smaller package, facilitating development of physically smaller vehicles with similar functionality.

1.3 Challenges of Vision-Based HMAVs

Although there are many groups working on similar problems, there are some universal challenges that limit the rate of development. First are the inherently unstable, fast, and nonlinear dynamics of HMAVs. This means that, in addition to being hard to control in a classical sense, when things do go wrong, they go wrong very quickly. Small or infrequent errors can have disastrous consequences, causing the vehicle to physically crash, potentially damaging itself, the onboard electronics, and endangering the people nearby. Even if replacement parts for these inevitable crashes are budgeted, the time necessary for frequent repairs takes away from the core research and reduces the freedom to take risks, which is important for progress.

Small vehicles also have small payload capacities for sensors and computation, severely limiting the data available to the system, the amount of processing that can be done onboard, and communication performance. Laser scanners are near the limit of payloads for HMAVs, and even though cameras are becoming increasingly miniaturized, the related computational requirements are significant and difficult to service within the payload limitations. Sensor data can be transmitted wirelessly to powerful offboard computers, but this introduces several more challenges with bandwidth, latency, and reliability of these wireless connections.

Image processing is a notoriously hard problem, limited physically, algorithmically, and computationally. Significant progress has been made in the last 30 years, but the state of the art is still far from the capabilities of human vision, which we often take for granted. Changing light levels, low indoor light levels, and tight payload requirements in size and weight are very challenging to deal with at the hardware

level, often causing poor exposure and image blurring. If good images are acquired, the challenge shifts to the algorithms that need to make sense of these arrays of colors, which could represent any combination of trillions of possible objects, actions, and environments, and may be moving, distorting, reflective, transparent, and so on. The state of the art in the field of computer generated imagery is nearing photo-realism, and can handle all of the physical features mentioned before quite well, but we are addressing the inverse problem, which is significantly more difficult. Not only is the data incomplete (3D data describing a 4D world), but the processing must happen in real time. Even with advances in embedded electronics, this real-time limitation forces most systems to work with low resolution, low frame rate, gray-scale images. As a result, significantly less data can be processed than would be available from, for example, a consumer High-Definition camcorder.

Finally, the added third-dimension of maneuverability and constraints makes environment mapping and path planning fundamentally more complex than typical 2D systems. Most obstacle avoidance schemes address the 2D case of a vehicle maneuvering through a pseudo-planer world, but in order to fully utilize the capabilities of an HMAV, it is important to allow any feasible trajectory. It is also important that the full 3D location of objects in the world be known so that they can be planned around according. In addition to complicating the algorithms for mapping and planning, adding a third dimension also increases the computational complexity significantly.

Although daunting, these challenges are surmountable in developing the capabilities sought. In the next section we will discuss progress made by other groups in this field, and in sections 1.5 and 1.7 we will outline our approach and contributions.

1.4 Background

The core technologies of this system fall into five categories: estimation filtering, image processing, environment mapping, reactive obstacle avoidance, and proactive obstacle avoidance, although we will only cover those most uniquely contributed to in this thesis in depth. There has been much research in all of these areas, and the

following sections focus on the most recent developments that have been used in this system.

1.4.1 Mobile Robot Systems

Mobile Robot Systems have demonstrated the utility of a wide range of sensor types, including LIght Distance And Ranging (LIDAR), SOund NAvigation and Ranging (SONAR), Inertial Measurement Units (IMUs), wheel encoders, RAdio Distance And Randing (RADAR), infrared ranging, and Global Positioning Systems (GPS) [43, 48, 67]. The most notable display of these sensor rich, highly capable vehicles is from the DARPA Urban Challenge event in which several teams competed with autonomous automobiles capable of sensing and processing the surrounding dynamic environment and planning through the resulting information in complex and robust ways [14]. MIT demonstrated a particularly complex entry with Talos, a Land Rover with fifteen radar units, six cameras, twelve LIDAR units, and a high-powered, forty-core processing system. The entire system was self-contained, including an additional gas powered electrical generator and air-conditioning system to maintain the proper environment for the electronics [40]. Brunskill, Kollar and Roy have also demonstrated advanced mapping and localization using only LIDAR data on a mobile robot [6]. On the other end of the spectrum, Davison, Reid, Molton and Stasse have addressed the Simultaneous Localization and Mapping (SLAM) problem from a wearable computing perspective, and have demonstrated real-time SLAM using only a single camera, but require known landmarks to begin within view of the camera to initialize the system [11].

There are several MAV and HMAV systems currently under development addressing the autonomous flight problem. There are many different combinations of vehicles, capabilities, sensor sets, and operating environments being explored. The three we will address here are collision avoiding fixed wing MAVs, advanced vision processing on helicopters outdoors, and vision-aided flight on indoor helicopters.

Forward flight MAVs have demonstrated robust flight stability and collision avoidance in small flight spaces. Often used as platforms for development of optical flow

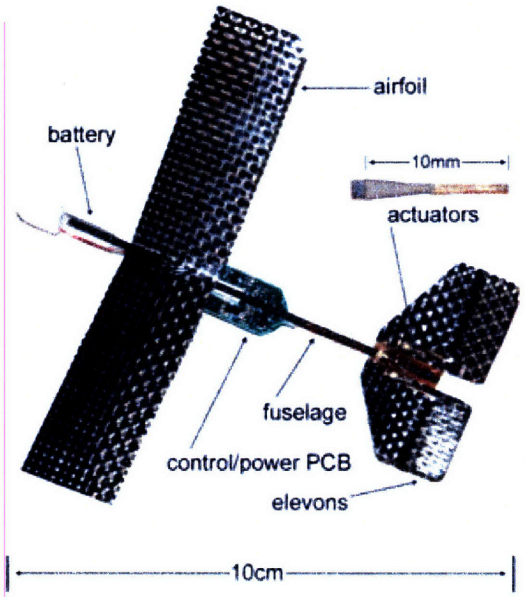


Figure 1-5: UC Berkeley 2 gram autonomous glider [68].

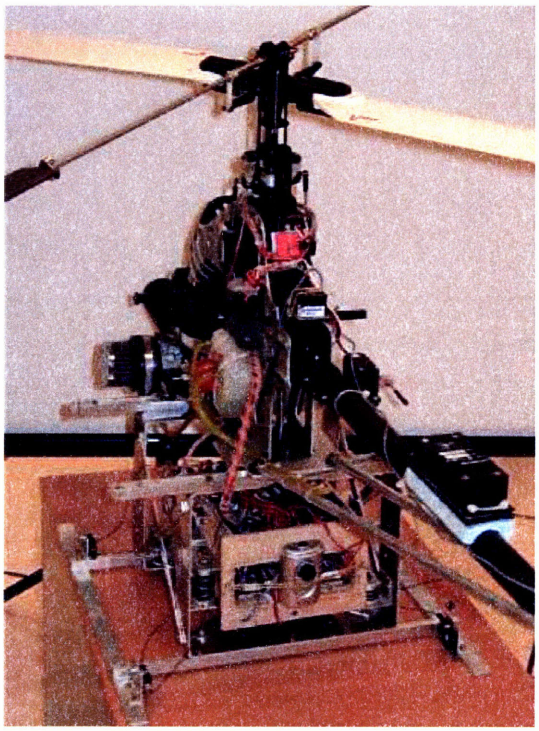


Figure 1-6: A typical vision-equipped autonomous helicopter [8].

based obstacle avoidance algorithms, they are typically designed to be dynamically benign, thus do not depend on accurate state estimates in order to fly stably. In order to stay safely airborne indefinitely, the primary challenge of the navigation system is to steer the vehicle left or right to avoid obstacles and approach targets, and change throttle or pitch to maintain altitude, neither of which require 3D environment mapping or an accurate position estimate [52, 56, 75]. Autonomous take-offs and landings mimicking the landing behavior of bees have also been demonstrated [21]. UC Berkeley has pushed the limits of these simple and lightweight sensors and electronics to develop a fully autonomous glider less than 3 grams, shown in Figure 1-5 [72].

Advanced vision-based ego-motion estimation on HMAVs, however, requires significantly larger and heavier sensors and computers, thus is typically done on large helicopters, such as the one shown in Figure 1-6, in wide-open, outdoor environments [27, 39, 62]. These platforms are often hand-piloted to collect data which is post-processed afterwards [13]. Several groups have developed helicopter platforms capable of estimating the vehicle position using only vision and inertial data

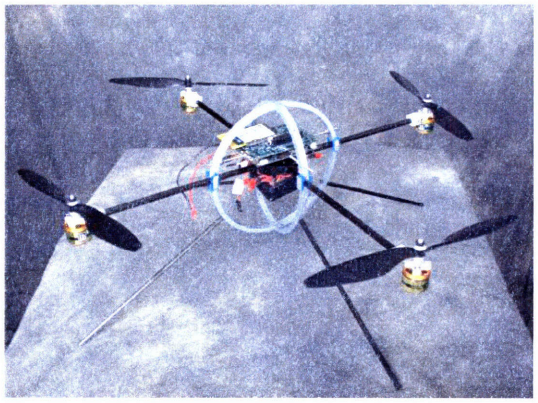


Figure 1-7: The quadrotor platform at the Robotic Vision Lab of Brigham Young University [18].

Figure 1-8: A commercially available quadrotor platform from AirRobot, GmbH [1].

using known visual targets that must be within view of the camera for the entire flight [8, 73]. Kanade, Amidi, and Ke have developed a more general vision-based ego-motion estimation scheme which operates on an autonomous helicopter flying outdoors under GPS-based control [32]. They have also demonstrated high-fidelity 3D terrain mapping with a LIDAR system onboard the vehicle. Several groups have also addressed the problem of vision-based target tracking and mapping for vehicle guidance, but still rely on standard GPS systems for ego-motion estimates [35, 59]. Saripalli and, however, have embraced the benefits of high performance inertial units, high resolution binocular cameras, and Extended Kalman Filter formulations similar to those employed here, achieving navigation estimates to within 1% accuracy, even after covering nearly 400 m of terrain, but again, the estimates are only done in post-processing after human piloted flights [33]. These results are promising, but there are still no results indicating these vehicles cannot close the control loop without GPS data or human pilots, and cannot be flown at all inside confined environments. Although much of the mathematics and hardware are similar between these systems and the one presented in this thesis, the environment is significantly different. Visual feature tracking in particular has significantly different behavior because of the nature of the confined environment, which limits the scope of the camera, and has a higher degree of clutter and complexity, increasing occlusion, which significantly reduces the persistence of features, increasing estimate drift.

Most HMAV research in indoor environments uses downward looking cameras to provide adjustments to slow lateral drift, caused by imbalances in the vehicle and disturbances from the environment, so the vehicle can autonomously hold position. The quadrotor MAV developed in the Robotic Vision Lab at Brigham Young University, shown in Figure 1-7, is an excellent example of this capability [18]. Features are detected on the ground and tracked between frames with a feature tracking algorithm implemented on a Field Programmable Gate Array (FPGA). As the features are tracked, their movements can be used to estimate the motion of the vehicle and a reaction is computed to compensate for the movement, keeping the vehicle hovering over the same location. The authors claim image processing can be done on the FPGA at greater than 15 frames per second, demonstrating the computational feasibility of image processing onboard an HMAV. AirRobot has also developed a commercial product, shown in Figure 1-8, capable of drift-free autonomous hover in unknown indoor environments [1]. This feature makes a very stable and easy to fly vehicle for human-in-the-loop control, allowing the operator to take her or his hands off the controls while the vehicle hovers in place, even in confined indoor spaces. Although very capable in the hands of a human pilot, neither is capable of estimating its absolute position while traversing unknown environments, cannot map obstacles, and thus cannot navigate unknown environments autonomously.

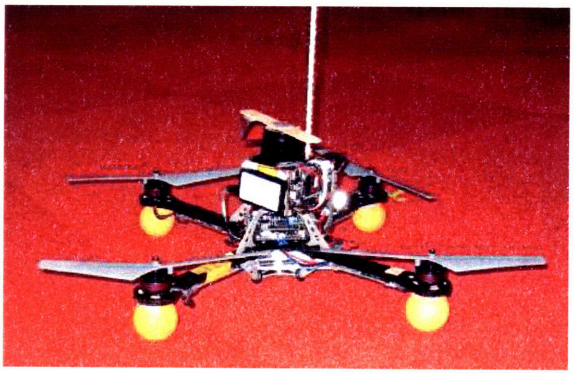


Figure 1-9: A LIDAR enabled quadrotor capable of estimating its position and heading by matching real-time measurements with a map of the room. [57].

He, Prentice and Roy have demonstrated the utility of LIDAR onboard HMAVs for position estimation in indoor environments, shown in Figure 1-9. These systems show promise for HMAV development with the continuing miniaturization of LIDAR modules, but so far require maps of the environment to enable registration of measurements, and poition estimates are too noisy to accurately estimate velocity for effective control feed-

back [57].

1.4.2 Obstacle Avoidance

Collision avoidance is an essential capability of almost all autonomous vehicles, and has been studied extensively on ground, air, and underwater platforms. There are three basic types of obstacles avoidance schemes: optimal, thorough search, and reactive.

Optimal schemes, such as Mixed Integer Linear Programming, typically optimize paths over several parameters, including fuel usage, obstacle proximity, minimum time, and other constraints specific to particular scenarios. Although provably optimal, these schemes often have difficulty dealing with dynamic and uncertain environments, and are often very computational intensive. They can, however, be effectively combined with other planning methods to balance optimality with computational tractability [16, 37, 54, 61].

The most commonly used thorough search type planning method is the Rapidly-exploring Random Tree (RRT) [58]. The basis behind an RRT is that the configuration space of a vehicle is significantly limited by its dynamics, thus random sampling of control inputs and simulated dynamic propagation will necessarily generate dynamically feasible paths which can be evaluated for merit. RRTs are very powerful because they scale very easily to fit whatever computational resources are available, and are very adept at working with complex and dynamic environments. Although technically they can scale to match the resources available, performance drops as computation is reduced and typically require large computational resources to do an adequate job [36, 38].

The primary method discussed in this thesis is reactive obstacle avoidance. These schemes can guarantee safety with minimal computational effort, but are prone to inefficient path generation and have a tendency to get stuck in local minima with no guarantee of ever finding a path to the goal [5]. There have been many approaches to this class of planning, such as the use of artificial potential fields [34], vector field histograms [4], and global dynamic windows [5]. With potential field planning, the

environment is modeled as a series of attractors and repulsors that interact as the vehicle moves through the environment [23]. These schemes have particular difficulty with head-on collisions where the vehicle tends to "bounce" off the obstacle before going around it. Attempts to fix this problem have been made by introducing curl into the potential field, causing the vehicle to sweep to the side of obstacles rather than repelling directly away from them, but these solutions often add significant computation to solve the necessary differential equations, and often generate non-intuitive paths that go the long way around obstacles [47].

1.5 Approach

In order to achieve the capabilities outlined in section 1.1, the vehicle must have sensors to estimate the vehicle state for control, as well as perceive the environment for obstacle avoidance. Because of the nature of environments we are considering, robust communication cannot be assumed, and the vehicle must also be capable of recovering from communication outages gracefully using an entirely self-contained system.

We have chosen to use inertial measurements and a single camera for our sensor suite. Inertial Navigation Systems (INS) have proven utility in a wide range of applications, and Inertial Measurement Units are becoming increasingly small and low-power with the advent of Micro-electromechanical Systems (MEMS). Video is the output data of choice for many missions, thus it is logical to use the same camera for robotic perception of the environment, as well as an improved ego-motion estimate sufficient for autonomous flight, eliminating the need for any additional sensors. It is also of note that we have oriented our camera forward-looking, allowing the detection of obstacles in the forward flight direction, but which also gives a more natural view of the environment for human interpretation.

Many impressive results have been achieved with GPS, but it is unreliable in many of the scenarios we are considering. GPS data can be combined with the system when the vehicle is flying outdoors, removing drift in absolute position, but the vehicle must

be fully-functional when GPS is not available. In this thesis, we demonstrate that our sensor set is adequate for autonomous flight of HMAVs in unknown environments. Nevertheless, there are fundamental limitations of our approach which are discussed in section 1.6.

The vehicle must also be able to plan in the face of an environment whose representation is in constant flux, typically due to the noise and other uncertainties inherent in physical systems. MIT's entry in the Urban Challenge was an excellent demonstration of these advanced planning capabilities, and used grid map based RRTs to constantly re-plan near-optimal trajectories at a high rate [38]. Although effective, these algorithms require orders of magnitude more computational power than is available onboard MAVs at this time. Fortunately they have been proven tractable on offboard computers which can communicate wirelessly with the HMAV, but the poor communication performance in the environments we are targeting makes it impossible to rely solely on these offboard planners.

We have designed our architecture to combine an onboard reactive planner with the MIT RRT planner running offboard, outlined in Figure 1-10. The resulting system allows the vehicle to navigate safely when communication fails, but also plan very intelligently while in communication range.

From a hardware perspective, our approach is to use as little custom hardware as possible, keeping the vehicle and sensing system simple so that we can risk crashes and easily replace broken equipment. An extension of this is our choice to run our algorithms offboard on a regular desktop with dedicated high-performance wireless links to the vehicle. This choice keeps the vehicle small, simple, and low-cost, but also gives more flexibility with developer tools and real-time output. Transmission of hardware IMU data proved to be too unreliable for robust flight, so we have chosen to simulate our IMU data with the Vicon tracking system which does not suffer from wireless dropouts. The Vicon tracking system also provides truth data for both post-analysis of system performance and a safety net which can be engaged in the event of poor algorithm performance during development, reducing the likelihood of crashes. The simulated inertial data is shown to have characteristics similar to its hardware

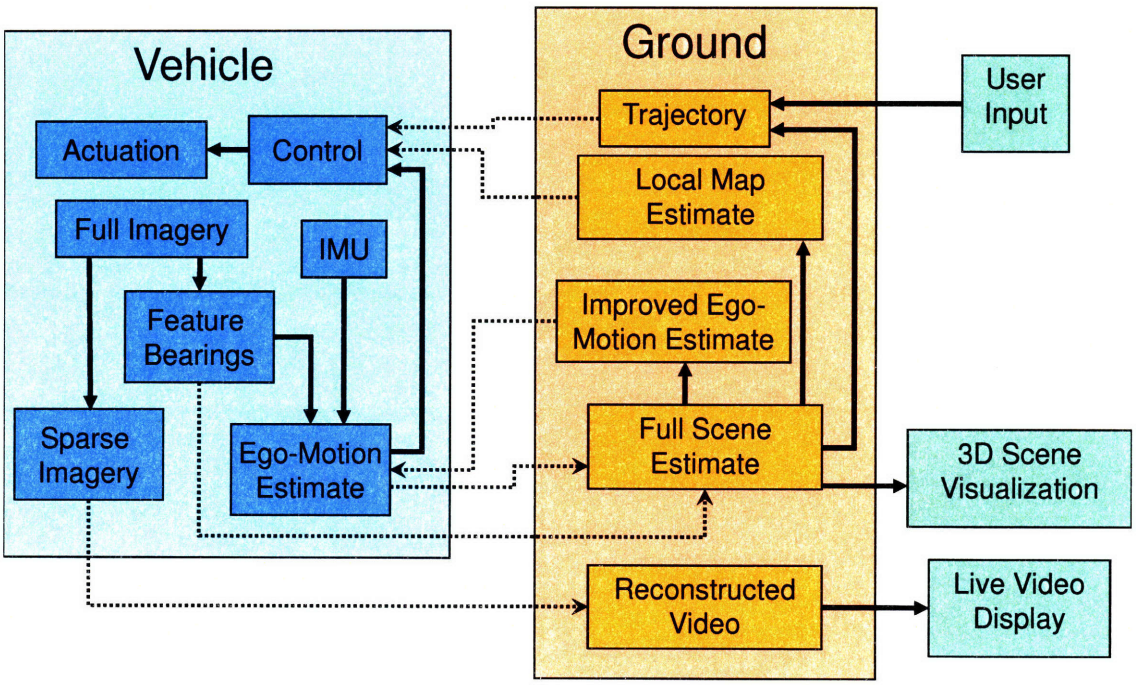


Figure 1-10: System architecture.

counterpart in section 2.2.4, and camera performance is thought to only increase with an embedded system, as latency and compression artifacts from JPEG compression and wireless transmission will be removed.

The solution we present in the following chapters addresses all these issues: autonomous planning and control in unknown environments aboard a hover-capable unmanned aerial vehicle using a six degree of freedom IMU and a single camera via a hybrid onboard/offboard planning system. Our estimation and control is demonstrated to be stable and capable of tracking paths with sufficient performance to avoid obstacles perceived in the environment, and the planning system can intelligently navigate unknown environments and keep the vehicle safe in the face of communication outages.

1.6 Limitations

There are many practical and theoretical limitations to the sensor suite we have chosen. The most obvious is that it does not work in the dark. There are not many

good solutions to this problem in arbitrary situations, but active illumination and infrared sensitivity show promise. SONAR and LIDAR are also strong candidates for perception in dark environments. Smoke, dust, and other particulates pose a serious challenge to any indoor flight, but infrared imaging, RADAR, and SONAR may present solutions.

It is also difficult for the system to track moving objects because of the single camera. The parallax necessary to measure depth is only provided by vehicle movement, which takes time, and thus the environment must not change between measurements. Figure 1-11 shows a scenario in which a static object (blue circle) and a moving object (green circles) are perceived by a moving vehicle with a single camera (orange squares). The lighter shades indicate previous locations, the solid arrows indicate movement, and the dashed arrows represent bearing measurements from the vision system. The two bearing measurements for the static object intersect at a point that coincides with the actual location of the object, thus creating an accurate measurement shown by the smaller blue circle. The bearings for the moving object, however, intersect above the object because it has moved from when the first measurement was taken, and thus its location cannot be accurately determined. This limitation can be overcome with binocular vision, which provides two bearings with parallax at the same instant in time for every obstacle perceived in the environment. It is also possible to use optical flow to compute time-to-impact which can be used to form an actionable understanding of the danger [52, 56, 75], but this cannot generate a physical mapping which is far more useful for intelligent planning.

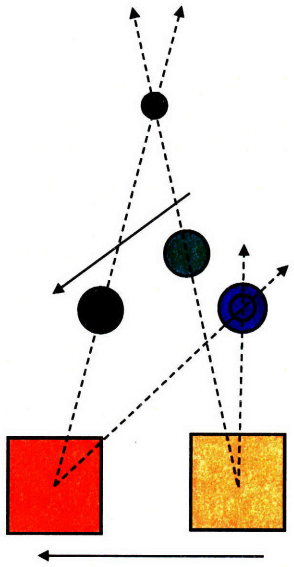


Figure 1-11: A diagram of dynamic single camera object estimation.

The specific camera hardware utilized in this work also limits the flight envelope of the vehicle. If the vehicle moves too quickly, the imagery becomes blurred beyond usability, and estimates of the environment and vehicle state both degrade. The

narrow field of view also limits the system performance by reducing the span of the surrounding environment that can be seen at one time, decreasing the persistence of features in the environment and hindering the mapping and ego-motion estimation performance. These issues are easily solved with a faster shutter speed and higher frame rate camera equipped with a wide-angle lens. These are currently available and becoming more prevalent, but were not in a convenient, digital, wireless, off-the-shelf form at the time of this project.

Finally, although the similarities of the simulated and hardware inertial data are shown, the vehicle cannot fly outside of the RAVEN flight space. Efforts in embedded hardware development will need to be made to implement the algorithms developed in this thesis on a fully self-contained onboard system in order to fly in real-world environments. Computational resource analysis is performed in section 5.2, indicating that the algorithms should be able to run on COTS embedded electronics.

1.7 Contributions

The primary contributions of this thesis are the system and key supporting algorithms for the guidance and control of an HMAV in unknown indoor environments. This includes the implementation and use of a hardware system that provides the sensing, computation, and vehicle for rapid prototyping, dramatically increasing the rate and productivity of development. The concepts and methodologies of the algorithms used for image processing and ego-motion estimation are discussed, and the implementations of the environment mapping and reactive obstacle avoidance algorithms are presented. Flight results of the autonomous system are analyzed, demonstrating the viability of this solution to achieve its objectives in real environments on physical hardware.

Chapter 2

System Design and Methodology

2.1 Architecture

There are many components that must interact in harmony for a complex planning and control system to operate effectively in a real-time environment. In particular, the onboard system must run very quickly and reliably in order to maintain control over the unstable vehicle, while the offboard system must be able to provide information that is useful to the onboard system, even when the data is asynchronous and unreliable. Communication between the modules is especially important on each individual computer (since each is running multiple threads) as well as between computers.

2.2 Hardware and Supporting Technology

The MIT Real-time indoor Autonomous Vehicle test ENvironment (RAVEN) [26, 69] provides the hardware foundation for the developments presented in this thesis. RAVEN combines the tracking capabilities of a Vicon-MX camera system [70] with R/C control hardware, digital video cameras, and an array of networked servers into an ideal development environment. Although ultimately the external tracking system will not be necessary, the vehicle is designed to fly only inside the RAVEN area so that the Vicon truth data can be used for post-analysis, as well as a control safety net should the algorithms under development become unstable during flight.

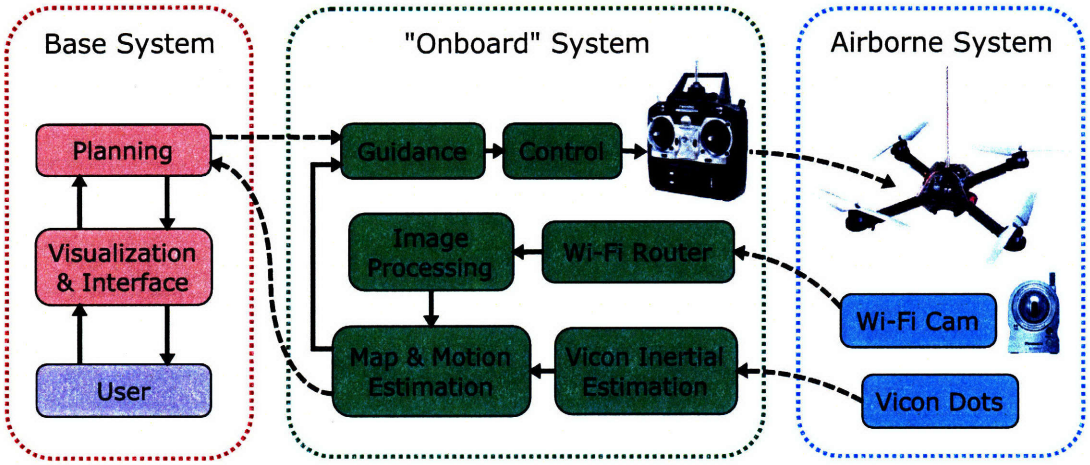


Figure 2-1: Hardware architecture and connectivity.

For ease of prototyping and development, as much computation is done offboard as possible. This keeps the dollar value of the airborne system low, allowing more freedom to take risks when conducting experiments. It also allows the use of COTS hardware, alleviated the time that would be required to develop custom vehicles and electronics. The computers use standard Linux development environments, which makes it very easy to develop, implement, and debug software quickly. The additional computational power and storage space compared to an embedded platform also allows many debugging freedoms, such as the ability to save very large data files, generate complex visualizations for real-time debugging, and playback rates significantly faster than real time for testing algorithmic improvements on the real flight hardware. Extensive process timing analysis is also done in order to analyze the feasibility of the core algorithms to ultimately run in real-time on an embedded system.

The basic hardware structure is outlined in Figure 2-1, showing how the system is broken down into four parts: a truly onboard system (outlined in cyan on the right), the conceptually onboard system (outlined in dark blue), the offboard base station (outlined in orange), and the communication links between them (dashed arrows). The truly onboard system includes everything that actually flies in the air, including the vehicle itself, the wireless camera, and the reflective Vicon dots. The conceptually onboard system is composed of a regular desktop computer[1], an 802.11g Wi-Fi router

[1]A powerful Dell OptiPlex with Core2 Quad Q6700 processor at 2.66 GHz and 4 GB of RAM is used so that it can effectively run the onboard system, offboard system, and visualization simulta-

for communication with the onboard camera, and the Vicon system for truth and emulating IMU data. The offboard system is composed simply of a separate desktop computer workstation.

2.2.1 Communication

Between physical computers, the Lightweight Communications and Marshalling (LCM) system is used to transfer messages and data over gigabit ethernet (or localhost, in the case of offline playback) [15]. LCM is a high speed and lightweight UDP based networking package that enables low latency, high bandwidth communication between multiple processes on a network without a central hub. An LCM communication manager runs in an independent thread on each machine and asynchronously calls handlers as relevant packets are received. The onboard system listens to commands such as a standby, takeoff, waypoints, and land, and the base station listens to information coming back from the onboard vehicle in the form of state and mapping estimates.

Because the communication framework is so modular, it is easy to swap in new message sources. To supply the base station with commands for the onboard vehicle, a 3D user interface has been developed so an operator can visualize the estimated map of the environment, as well as direct the vehicle with a point and click interface. The planning software from the MIT Urban Challenge can also be plugged into the system – the base system processes the map estimate and sends it to the planner so that it can plan through the perceived environment, and with a slight modification, the plans are transmitted as waypoints to the base system, which then sends the appropriate commands to the vehicle, just as if an operator had clicked the screen.

Having a unified communication system also makes it easy to simulate degrading communications due to the physical environment. For example, if the vehicle flies behind an obstacle known to the simulation system, the communication performance can be decreased accordingly, enabling testing of the system response to communication failures.

neously when conducting offline playback tests.

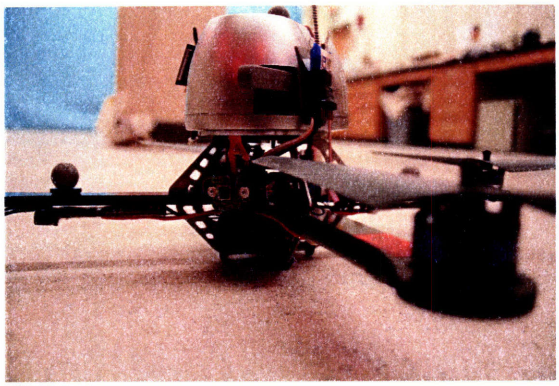


Figure 2-2: Closeup of quadrotor with camera attached.

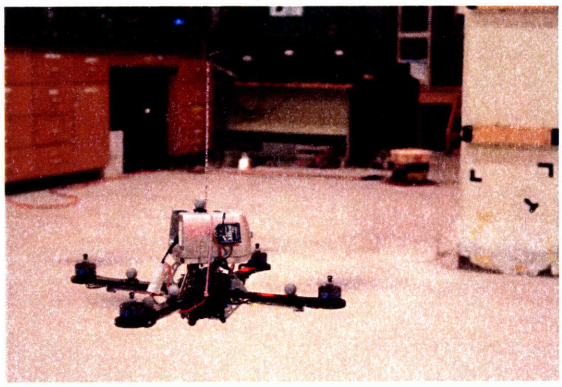


Figure 2-3: The quadrotor with camera and radios in flight. Flight computers and operator station in background.

2.2.2 Vehicle

The HMAV chosen for this research is the high performance Hummingbird quadrotor with the scientific accelerometer add-on, shown in Figures 2-2 and 2-3, part of the professional line of helicopters from Ascending Technologies [2, 22]. The vehicle is outfitted with an embedded electronics suite, including an 8-bit microcontroller (AVR ATMega8), digital to analog converter, three temperature compensated piezoelectric gyros, three accelerometers, and four brushless motors and custom drivers. The onboard controller is optimized for high-speed fixed-point computation, and runs at a rate of 1 kHz. This high speed means that the sensors are over-sampled much faster than their rated update rate of 50 Hz, giving the controller a better statistical measure of the true physical parameter, and the response time of the motors is minimized to a maximum 1250 μs from sensor measurement to torque change. This rapid response and high sensor fidelity gives the vehicle very high control bandwidth, making the platform very stable in flight. The accelerometer and gyroscope data is also fused to generate a tilt estimate which is used for the onboard control, thus the external controller need only specify the desired tilt angle, and the vehicle will match this angle with very high performance. In low-disturbance environments, a well trimmed vehicle can maintain nominally level flight and simply drift slowly around with ambient air currents.

The motors and controllers are high efficiency brushless types, which give the

vehicle a high power-to-weight ratio and are very efficient, extending the vehicle flight time beyond 30 minutes on a single Lithium-Polymer battery charge. The high power-to-weight ratio gives the vehicle enough agility that it can perform flips and other acrobatic maneuvers inside confined areas (guided by the commands of an expert human pilot), but also general control of the vehicle can be very aggressive without saturating the actuators, allowing the vehicle to rapidly avoid incoming obstacles or respond to sudden wind gusts without becoming unstable.

The frame is sturdy and light-weight, providing stiffness for control performance, strength to survive crashes, and light to maximize power-to-weight and flight time. The four arms are composed of carbon fiber and balsa wood composite laminates, joined by two central carbon fiber plates to which the electronics are mounted. A magnesium cage encloses the electronics on top and the battery below the central plates. The magnesium is stiff and strong, but also ductile, efficiently absorbing the impacts of crashes by deforming. The performance of the vehicle is not affected by deformations to the cage, which can be bent back into place by hand.

The vehicle is also outfitted with a digital communication channel, which allows external access to the gains, control commands, sensor readings, and tilt estimate. This functionality makes the system very easy to work with, and gives ultimate flexibility in the vehicle control, presenting a very high level of utility.

The vehicle receives control signals from the system via a standard 72 MHz R/C radio transmitter, which is plugged into the computer via a Tom's RC SC8000 buddy box [66] to provide a USB interface for the PPM signals necessary to control the vehicle through the trainer port on the transmitter. This interface provides a low latency and robust control link to maximize vehicle flight performance and reliability.

2.2.3 Camera

The camera system is a COTS Wi-Fi enabled network camera from Panasonic (BL-C131A), connected to the network via a 802.11g 54 Mbps Wi-Fi router. The camera hosts images which are retrieved through the network as JPEG files and loaded into the image processing system. The camera was physically modified for clean inte-

gration into the vehicle structure. The camera comes with an integrated pan and tilt system which has a small camera submodule, visible in Figure 2-4, that is connected to the rest of the camera with a flexible ribbon cable. After removing the pan and tilt motors, the small camera submodule was remounted cleanly to the vehicle frame with an adjustable mounting bracket for optimal alignment, and the rest of the camera body and electronics could be placed on top, directly above the center of gravity of the vehicle, keeping the dynamics symmetric and simple to control.

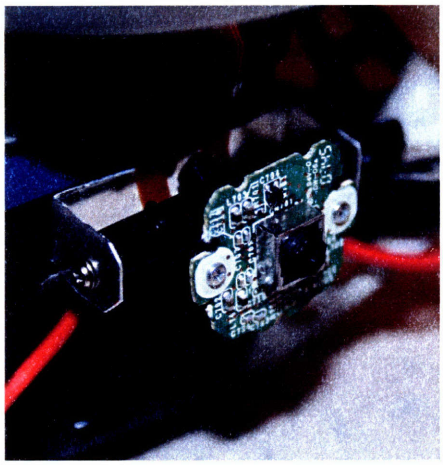


Figure 2-4: Closeup of camera with mount and ribbon visible.

The camera specification claims it can serve 640×480 pixel images at a rate of 15 Hz, but empirical testing shows the best operation is at 320×240, which runs between 15 and 20 Hz because of the low indoor light level. The lower resolution is fine for our needs as the image processing must be as fast as possible in order to run on an embedded platform, and the fewer the pixels the faster the processing. The 1 mm diameter lens has a field of view of 41 degrees vertically by 53 degrees horizontally and a fixed focal length of 20 feet to infinity. The small lens, long focus, and narrow field of view make the optics very close to an ideal pinhole camera, indicating rectification is unnecessary, simplifying the image processing. At the same time, the narrow field of view limits the vehicle's ability to perceive the environment, making it more likely to blunder into unseen obstacles, and also reduces the persistence of features, decreasing ego-motion estimation performance as discussed in section 3.2.

The usage of a digital wireless camera has been essential to the success of the project. Analog radios in indoor environments suffer from multi-path interference, causing almost unusable image quality, especially because of the sensitivity to noise of machine vision. Wi-Fi is very well suited for indoor environments and provides low latency and excellent transfer rates, and the digital nature of the images means there is no randomized transmission noise to interfere with the image processing. Although

JPEG compression artifacts do have a noticeable effect on image quality, they are much more consistent than analog noise, and the frame-to-frame style processing we are using is largely unaffected.

2.2.4 Inertial Data

There are two options presented for measuring inertial data of the vehicle. The first method developed is based on the robust and adaptable Vicon camera system which can provide high fidelity inertial data for any vehicle tracked in the RAVEN room. This is a very useful resource because it means that any vehicle, or even object, can have the equivalent data of a high performance onboard IMU without any hardware integration time, costs, or headaches. These differentiations are calculated by means of 2nd and 1st order extended Kalman filters on the position and attitude data, respectively, coming from the Vicon system, implemented with tools from the KFilter library [74]. A gravitational component is added to the resulting acceleration estimate, which is rotated into the body frame of the vehicle. Finally, Quaternions are computed, and the full data set is broadcast on the data bus as LCM packets, emulating a hardware IMU operating at 100 Hz onboard the vehicle, as well as providing a truth reference.

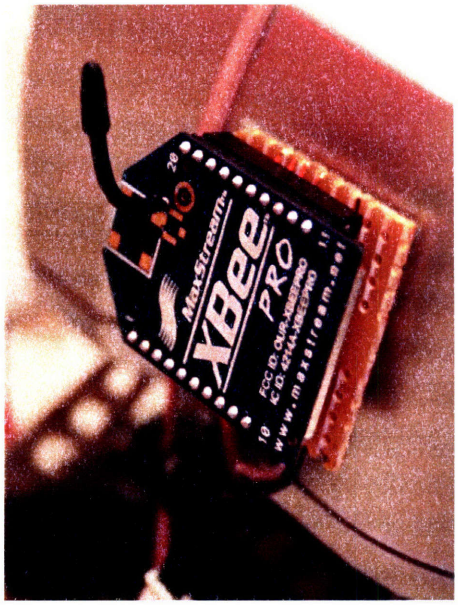


Figure 2-5: Closeup of XBeePro digital radio for connection to hardware IMU.

The second option is to access the hardware IMU data onboard the vehicle via a digital radio. As shown in Figure 2-5, an XBeePro digital radio is interfaced with the high performance X-3D-ACC quadrotor sensor board to transmit hardware IMU data directly to the "onboard" ground computer. The transmission has little latency and is very close to a true embedded flight system because of the dedicated data link.

Although using hardware IMU data would be a closer approximation to a true

embedded flight system, occasional dropouts and corruption in the transmission has presented a challenge in using the data offboard. The ego-motion estimate and unstable vehicle dynamics are not forgiving to dropouts in data when it is expected to be updated at a consistent 50 Hz, and the vehicle could not be flown in this configuration. Additional filtering or other techniques could potentially address this problem, but Figures 2-6 and 2-7 show a comparison of the hardware IMU data to the filtered Vicon data, indicating that the simulated data is both more reliable, because there is no wireless data link (note the spikes and dropouts in Figure 2-6), and similar in characteristics to the hardware data, possibly even inferior, indicating that if the system can work with this emulated data, there is a good chance it would work with real hardware data on an embedded system.

The Root Mean Square (RMS) noise levels of the hardware IMU data[2] are 0.028 m/s^2 and 0.89 deg/s, slightly lower than those of the Vicon data, which are 0.037 m/s^2 and 0.96 deg/s. The simulated gyro data also has 40 ms more lag than the hardware data as a result of the filtering, which is an additional inferiority. The muted values of the simulated data are also apparent in the gyro plot of Figure 2-6, specifically from 24.5 to 25 seconds – this is a result of the filter reducing the noise in the system to match that of the hardware data, and thus attenuating the true data, indicated by the reduced magnitude of oscillations. The exact level of attenuation is driven by designing the filter to match the noise floor of the hardware data when the vehicle is at rest.

The primary disparity remaining between the two data sets is the heavy discretization of the IMU data, which is clearly shown by the wide gaps between bars in Figure 2-7. This discretization is due to the fixed precision of the embedded electronics, which evaluates to 0.020 m/s^2 and 0.61 deg/s in physical terms. However, because the magnitudes of the RMS noise levels and the operational values are larger than the respective fixed precisions, the discretization should not effect the performance of the filter significantly. Because of the close match of noise characteristics

[2]Hardware RMS values and histograms were generated with outliers in the hardware data removed, which are presumed to be caused by transmission errors and thus would not effect an embedded system

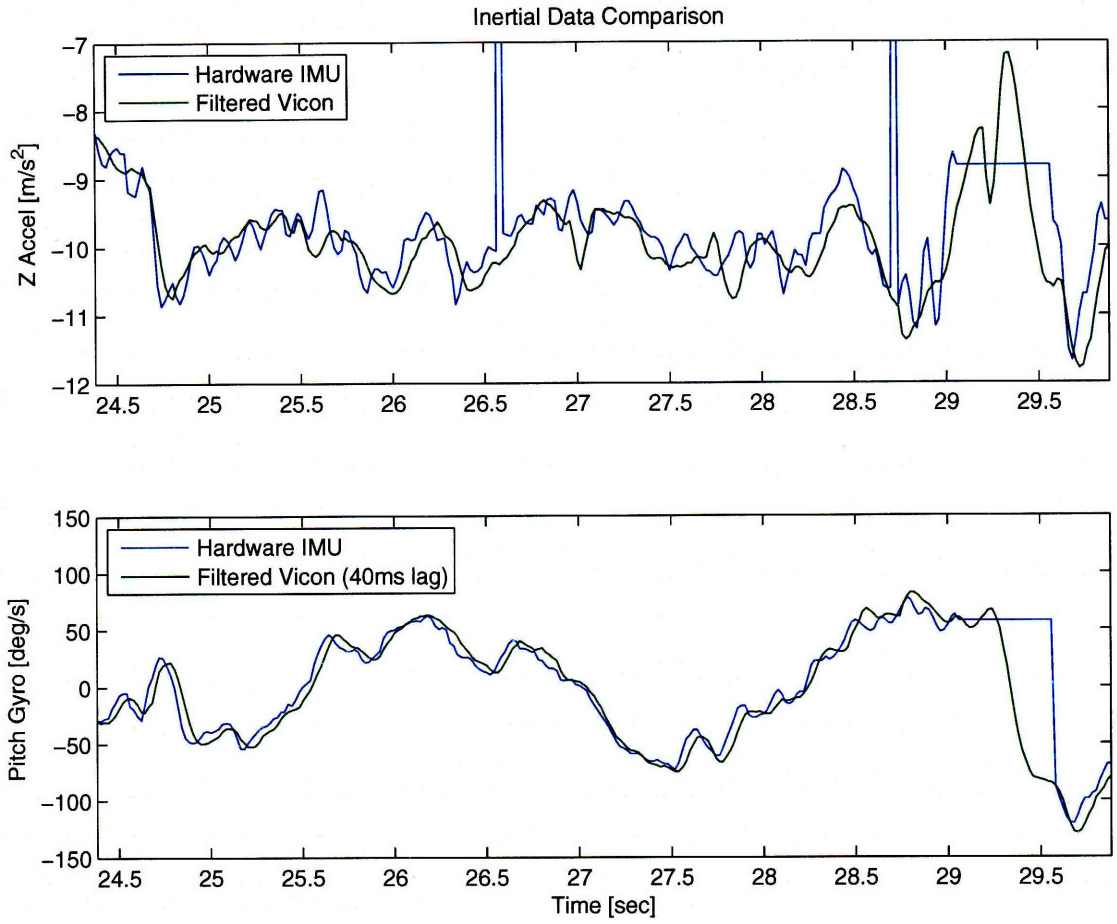
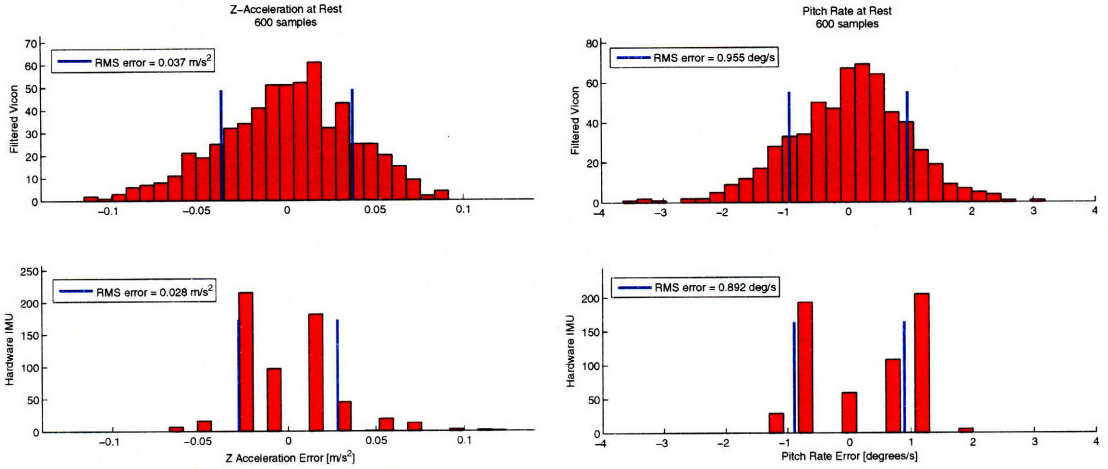


Figure 2-6: Comparison between inertial data measurement systems. Note dropout between 29 and 29.5 seconds.

and increased reliability, the simulated inertial data from Vicon was used for filter inputs on all flights, and it is suspected that performance would be similar, if not better due to the reduced latency and attenuation, when using real hardware data, which would not suffer from transmission errors or dropouts in an embedded system. Thus, it is hoped that the existing ego-motion estimation system would operate with similar, if not higher performance, when implemented on an embedded system.



(a) At-rest Z-Axis acceleration comparison. (b) At-rest pitch-axis rotation rate comparison.

Figure 2-7: At-rest inertial data comparison.

2.3 Onboard System

The onboard system needs to handle several tasks simultaneously, and thus interprocess communication is very important. The simplest solution to this problem is the use of buffers and a central hub configuration such that there is only one process that reads from any given buffer, and only one that writes to it. This decreases the overhead and delays associated with mutex locks and other techniques for guaranteeing thread safety. It does not, however, guarantee there will not be any thread racing conditions, but the system is arranged such that these conditions are minimized and that they are handled gracefully. There are a few low bandwidth situations where locks are more appropriate and they are noted below.

The onboard system consists of four independent threads: central hub, image processor, IMU reader, and communications handler, outlined in Figure 2-8. The communications handler is the simplest – it is spawned by the LCM communications system, and listens to the network, waiting for pertinent messages to forward on to the subscribed handler functions, which include commands for takeoff, land, add waypoint, and switch to Vicon safety net. Because the handlers are called asynchronously and at a low frequency, locks are held when updating the waypoint list. This is especially important because the multi-threading behavior of the C++ vector template

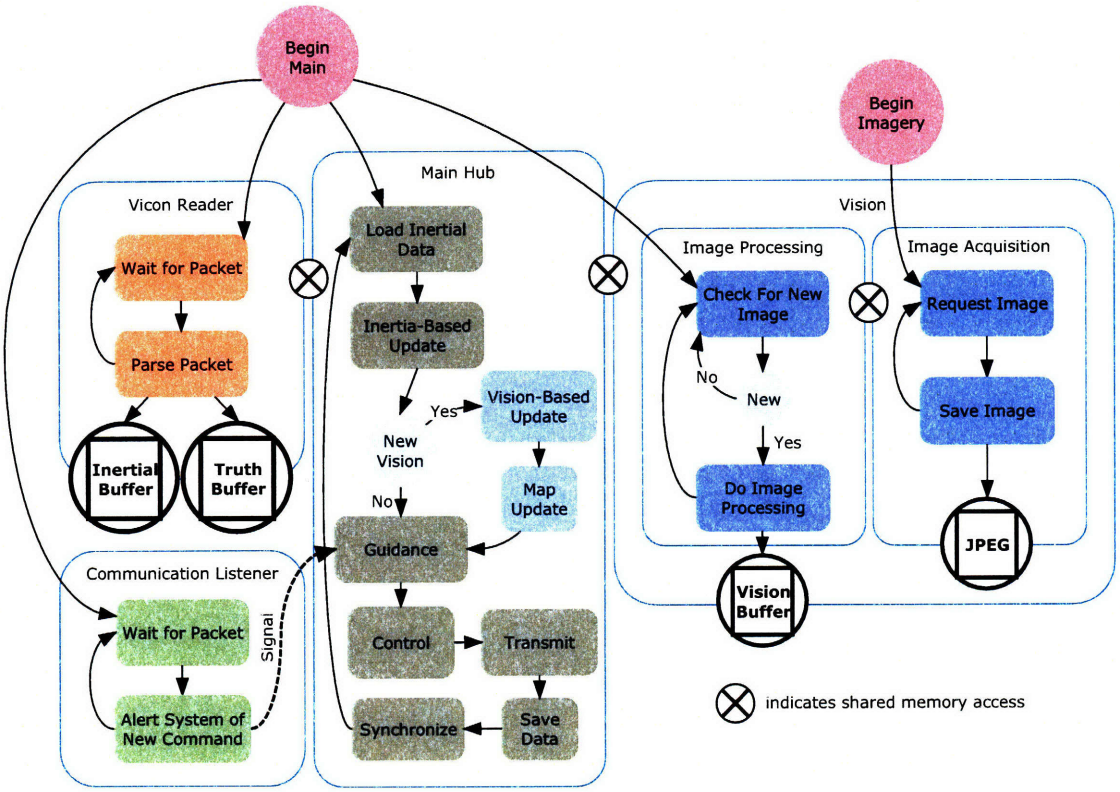


Figure 2-8: Process flow diagram of onboard system operation.

class used for waypoint storage is unknown, and if a read is called in the middle of a write, a segmentation error could result.

The image processor runs whenever there is a new image available from the camera, which are served by an independent image acquisition thread, at approximately 15 Hz, described in detail in section 3.1.1. After the image is processed (see section 3.1.2 for details), the new feature data is saved to a buffer and the data is marked as new. Because the central hub is operating at more than three-times this frequency, this flag is checked much faster than it is updated, and thus the data is read and used before it can be written again, minimizing the chance of thread collisions.

The IMU reader is clocked by the remote IMU source, either from Vicon or hardware) at about twice the frequency of the central hub, and thus uses a double buffer to separate reading from writing. As it receives raw data, it processes it into physically meaningful values and deposits them in alternating buffers. Once a buffer is written, the read flag is switched, indicating that the hub should read from this current data. If the hub happens to be reading the data at the exact time the next IMU data is

received, there is no collision because the data is written to the opposite buffer. If the read index is changed during a buffer read, the data will be split between two time steps, but because of the high rate of data this simply means that the most recent possible data is being used and does not cause any problems.

The central hub is clocked to run at precisely 50 Hz average rate using a timed semaphore. Its primary duty is to march through the system process and call each function in sequence. The first function called is the IMU updater, which loads in the most recent IMU data, as well as the most recent Vicon position data for truth comparison and potential safety net. The IMU data is fed into the ego-motion estimation filter, and if there is new vision data (which is true about every three frames) then a vision update is called as well, along with a call to the environment mapper. This generates an updated ego-motion estimate and an updated map which are subsequently fed into the onboard guidance system. Details of the motion estimator and environment mapping can be found in sections 3.2 and 3.3 respectively.

Once the system is armed with up to date knowledge of the situation, the guidance system examines the light state (standby, flying, landing, etc.) of the vehicle, the current waypoint list, the map, and the ego-motion estimate and generates a reference state for the control system to produce the desired motion, whether it is to move to a new location, avoid an obstacle, land, etc (details in section 4). The state estimate is then fed into the controller, which sums the products of the state errors and full state feedback gains to generate physical control commands for the vehicle. These physical commands are finally fed into the radio uplink function which converts the physical values to hardware commands for the vehicle, and transmits them over the serial port to the R/C transmitter that is flying the vehicle. The final acts are to save a new line to each of the data files, send data to the offboard base system, and yield the thread for the appropriate amount of time to maintain the 50 Hz average operating frequency.

2.4 Offboard System

The offboard, or base system runs the higher level algorithms, visualization, and operator interface software. The three primary elements are the 3D visualization and user interface, the communication manager, and the path planner. This system works in concert with the onboard system, providing guidance and structure to the reactive planning. Operating on a separate desktop, the system is designed to operate in the event of simulated communication failures. When communication is in place, the offboard system needs to present information in a way that fits in easily with the onboard system, without disrupting its operation. This is done by transmitting simple waypoints which specify a goal location and heading. These waypoints can be strung together into coarse, spatial paths that are interpolated by the onboard system, or only the last (or single) waypoint is considered and the onboard system is left to find its own way to the goal.

2.4.1 Operator Interface

The simplest embodiment of this high-level control is the point and click user interface. Figure 2-9 shows the visualization, powered by Visual Python [71], along with a control panel, raw data in a terminal window, and video streams from the onboard camera. The 3D representation of the world is rendered based on a sparse information set of estimated obstacle locations from the onboard system (the map), indicated by red ellipses, and hard-coded objects, such as the yellow block indicating the pole, for operator awareness while testing. The ellipsoids vary in dimension and darkness based on the confidence of the estimate. The larger and darker they are, the less confident the system is that the estimate is accurate. Both the true and estimated states of the vehicle are shown by the large and small quadrotor cartoons, and their path histories are indicated by the black and red traces, respectively. The reference point is shown by the blue orb, and the trajectory history is indicated by the thin blue trace. This data presentation allows the operate to judge how well the control, estimation, and environment mapping are all working in real-time, increasing the

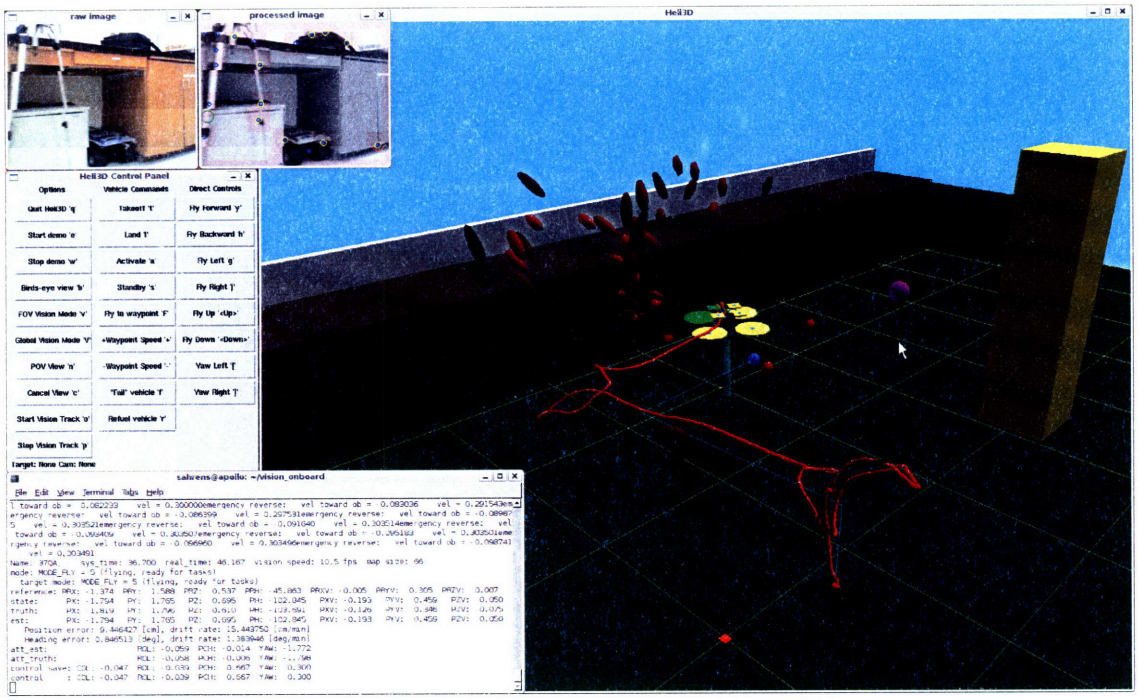


Figure 2-9: The 3D interface for point-and-click control shows true and estimated position, mapping, and control reference. Real-time vision and terminal output are also displayed.

speed of development compared with pure post-analysis.

The operator can also see relevant data in the terminal window, such as operation mode, position with millimeter resolution, and more. This information stream is updated at 1 Hz and serves as a useful tool for collecting information less easily depicted in the visualization, such as velocity and control commands. System messages are also displayed asynchronously, indicated the receipt of waypoints and other commands, as well as warnings of obstacle proximity.

Both the raw and processed video streams are displayed, showing the operator what the vehicle is seeing in real-time. The unprocessed, raw stream is useful for interpreting the environment, while the processed stream shows what the image processing is doing. The blue circles indicate features that are being tracked for environment mapping, and the yellow circles show the features being used for ego-motion estimation (note that all yellow circles also have coincident blue circles). Larger cyan, green, and red circles (not shown) pop up as features are discarded because of either large

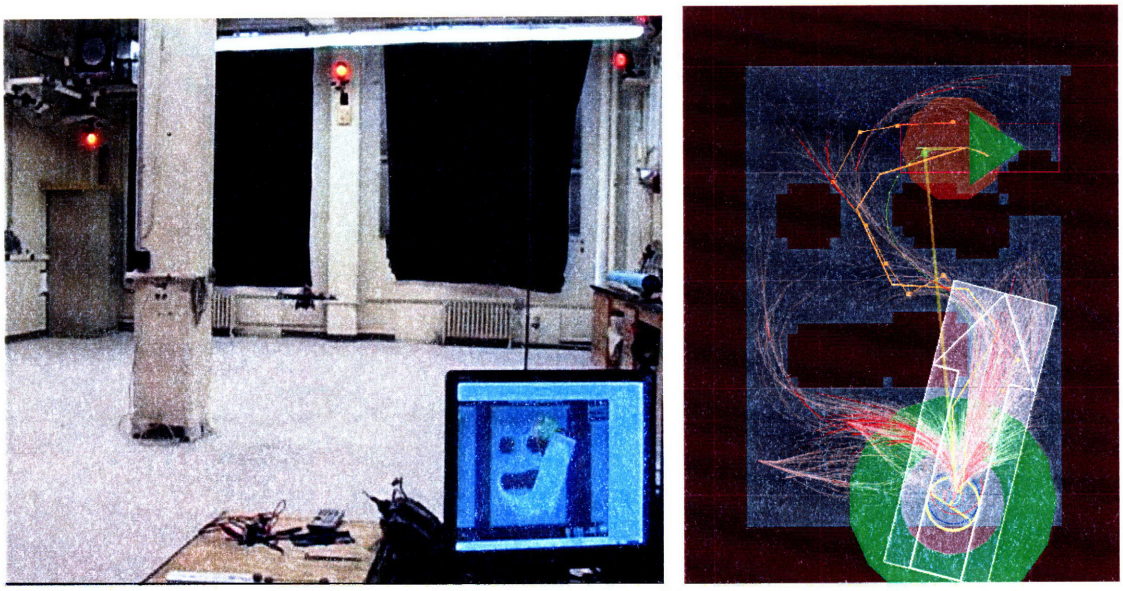
filter residual, tendency to clump together, or large jumps, respectively. The details of these feature culling behaviors are explained with more detail in section 3.1.2.

In order to specify a goal, the operator simply clicks a point on the ground plane to specify an x-y coordinate, which is combined with a desired altitude and heading to define the goal. The figure shows how the goal (indicated by the purple orb) appears directly above the cursor. The onboard system loads this goal as a waypoint and begins to generate and pursue a path toward that point. As obstacles are detected, the onboard system adjusts its trajectory to avoid them, doing its best to arrive safely at the goal. The operator can also use the keyboard or control panel to adjust the desired altitude and heading. As new waypoints are added, the old ones are erased and the vehicle pursues the latest waypoint immediately, allowing for simple corrections to the vehicle path as the operator sees fit. If communication is lost or the operator abandons the station, the vehicle will continue on to the last goal it was given, then hover in place until another command arrives, the battery dies, or another simple reaction is pertinent.

2.4.2 Data Playback

The system also has two playback systems. The first is a simple packet recorder that is built into the LCM package. Because all of the visualization information is communicated through LCM packets, the LCM packet logger can capture this information and play it back for later visual analysis. Although Matlab plots can also be generated, the 3D playback is uniquely beneficial because of the inherent complexity in visualizing the amorphous four-dimensional world that the vehicle operates in. With play, rewind, pause, zoom, and rotate commands, any time slice of a particular experiment can be inspected from any angle, showing estimation error, tracking error, and mapping performance in an intuitive way.

The second play-back system is specifically designed for rapid and controlled algorithm development. As the system runs, data files are recorded and saved, primarily for plotting and other post-processing. These data files can also be loaded by the onboard system itself to emulate the past experiment. The inertial data and way-



(a) Video screenshot of a quadrotor following the waypoints from the DUC planner.

(b) Screenshot of the planner operating on the static environment.

Figure 2-10: Preliminary results flying a quadrotor with the MIT Darpa Urban Challenge RRT planner.

point inputs are time-stamped against the saved images from the onboard camera, and loaded into the system as if they were being generated in real-time. The system processes the data as if the vehicle were really flying, so incremental changes to the image processing, ego-motion estimation, and environment mapping algorithms can be made and the effects can be seen immediately in the visualization[3]. During playback the system also runs as a single thread so that accurate measurements of process timing can be made for computational analysis, results of which are presented in section 5.2.

2.4.3 Advanced Autonomy

For autonomous missions, a similar waypoint interface is used, but the waypoints are strung together into a sequence of goal points (an option available to an operator

[3]Running data sets with no code changes produces results almost identical to those obtained during the original flight, but the slightly random nature of the multi-threaded real-time operation causes small discrepancies. If two data sets are both generated offline with identical code, however, then they will have identical results.

as well). Powered by the MIT DUC RRT planner, these waypoints are generated to form efficient trajectories through complex environments with many obstacles and constraints. The DUC planner was not a focus of this thesis, so the details will not be presented, but images from a preliminary integration of the planner with the quadrotor system are shown in Figure 2-10. The basic principle of the RRT planner is to sample the control space of the vehicle and propagate the vehicle motion through a controller and dynamics simulator to generate a trajectory, then evaluate the cost of the trajectory based on time, fuel, obstacles, and other constraints. Different samples cause the trajectories to separate into unique paths, forming a tree of control sequence options which can get quite dense, as seen in the pink curves of Figure 2-10(b). The cost evaluation is used to prune the tree, and the best trajectory is selected, indicated in the figure by the green line.

In the DARPA challenge, the control samples of the selected trajectory were then fed into the vehicle controller [36, 38], but in the case of the quadrotor, the propagated trajectory itself is sampled to generate waypoints which are transmitted to the onboard controller. The waypoints are spread approximately one body length apart, or about 30cm, in order to minimize transmission bandwidth, and are only updated a few times every second. This coarseness in both time and space is balanced by the reactive planner onboard the vehicle, which is always what generates the final movement plan. Having the reactive planner in the loop gives the vehicle the ability to avoid obstacles that appear without warning, even if the offboard planner has not had time to consider them, or if communication with the base station has been lost and the offboard planner is incapable of updating the plan. This implementation is in progress, and has not been completely implemented due to time constraints. For these preliminary results, the planning was done in 2D, the obstacles were hard coded, and the onboard obstacle avoidance was not active. The original car dynamics were also still being used as the propagation model, which actually proved to work quite well. The maneuvers considered, however, were artificially limited by not allowing lateral movements or in-place rotations.

2.5 System Design Summary

Although complex, this system architecture has been broken down into several independent parts that are capable of working robustly together to execute very complex demands. Emulated inertial data from the Vicon-MX camera system proves to be a very effective way to develop inertial navigation algorithms in an offboard testbed environment such as RAVEN. The performance comparison of a hardware IMU is also promising for the eventual migration to a self-contained embedded system that can fly free of the RAVEN flight room. A method for integrating high-level, computationally intensive planners with severely payload restricted HMAVs is presented, elaborating on the concept of a hybrid onboard/offboard scheme for keeping the vehicle operational when communication with the high-level planner is severed.

Chapter 3

Estimation and Mapping

As the vehicle moves through the environment, motion is evident in the changing camera images, and the IMU gives indication of accelerations and rotational rates. Using only inertial data, it is theoretically possible to perfectly estimate 3D vehicle motion, but noise in the sensing causes estimation errors. These errors are particularly troublesome because they are formed by the constant integration of noise and biases and thus can grow continuously. Using only inertial data, the vehicle would have no way to tell it was drifting into a wall at constant velocity. Visual feedback provides an indication of absolute position and attitude, but suffers from physical ambiguity—only dimensionless bearing angles can be measured reliably, thus the scale of the environment cannot be determined with only monocular vision information[1]. Binocular vision disambiguates this data with a known physical separation between two independent points of view, but we have chosen not to pursue this method in the interest of simplicity and miniaturization.

Tilt angle, essential to the stabilization of a hovering platform, can in fact be estimated using only inertial data, as demonstrated by the Nintendo Wiimote [20]. Because our vehicle is flying with fairly well known dynamics, its behavior relative to the orientation of gravity can be estimated with even greater certainty. Specifically, the tilt angle can be determined with bounded error, and thus the vehicle can be

[1]It is also feasible that scale can be estimated by object classification (for example, if the system recognized a doorway, it could assume a scale on which to base the rest of the world), but these techniques are very limited and require extensive computational resources.

maintained in a nominally level state, minimizing lateral acceleration and thus stabilizing the vehicle. Combined with GPS, a position stable system can be achieved, as demonstrated by several groups [2, 31, 64].

Inertial data on its own, or even combined with GPS, can still not effectively perceive the environment short of crashing into it, and thus an additional sensor package is necessary for collision avoidance. In our case, the vision sensor is utilized for environment mapping, while simultaneously augmenting the ego-motion estimate in the absence of GPS. Images on their own are not useful, however, and must be processed extensively, as discussed in the following.

3.1 Image Processing

Visual perception is essential to our system performance because it is the only way the vehicle is capable of sensing the environment, and it is a direct function of absolute position and orientation, unlike inertial data which only provides an indication of rates and tilt. The processing of these images relies on several algorithms and technologies, including image acquisition, feature tracking, feature selection, ego-motion estimation via fusion with inertial data, and ultimately environment mapping [32].

Some of the low-level technologies used in this system for the image processing are adopted from the Open Computer Vision Library (OpenCV) [28], an open source toolbox of common image processing algorithms used widely for development of a variety of projects, from mobile robotics to gene sequencing. The primary algorithms we have used are the pyramidal Lucas-Kanade optical flow calculator (`cvCalcOpticalFlowPyrLK` [30, 45]), and the good features to track feature finder (`cvGoodFeaturesToTrack` [63]). The details of these low-level algorithms are not of particular interest to this thesis, so we will limit the discussions of them in the following sections to the basic concepts, and instead focus on the higher-level, unique contributions.

3.1.1 Image Acquisition

The onboard camera is set up to connect to the WPA encrypted 54Mbps 802.11g wireless network hosted by our dedicated router. The router is connected directly to a dedicated ethernet card in the "onboard" desktop, minimizing packet collisions and the associated latency. Originally we tried to share the main data bus used by all of the computers in RAVEN, but occasional network issues caused unacceptable performance. A separate python script is spawned which makes a TCP/IP connection with the camera to request and download JPEG images from the camera as quickly as possible.

The JPEG images require decompression before they can be used by an image processing filter, which is implemented in the OpenCV function `load`. Unfortunately, the interface for this function is designed only to read images from the file system, and thus the images pulled off the camera cannot be fed in directly through a function callback or similar protocol. Because of the flexibility with which Linux presents it's file system, a portion of RAM can be mounted as if it were a physical drive on the computer, known as a ramdisk. The camera images are thus saved and loaded to the ramdisk as if they were files, but the performance is that of working entirely in memory. All other file processes, such as saving data files and writing videos, are also done on the ramdisk to minimize I/O stalls.

It is worthwhile to note that even though the mounted RAM file system is very fast, the JPEG decompression algorithm still requires about six milliseconds to decompress an image (plus the time the camera takes to compress and transmit it), which is a significant portion of our 60 milliseconds allocated per frame based on an average 15 Hz frame rate. In an embedded system, the microcontroller would have direct access to the raw pixel data, and no compression, decompression, or transmission would be necessary for onboard processing.

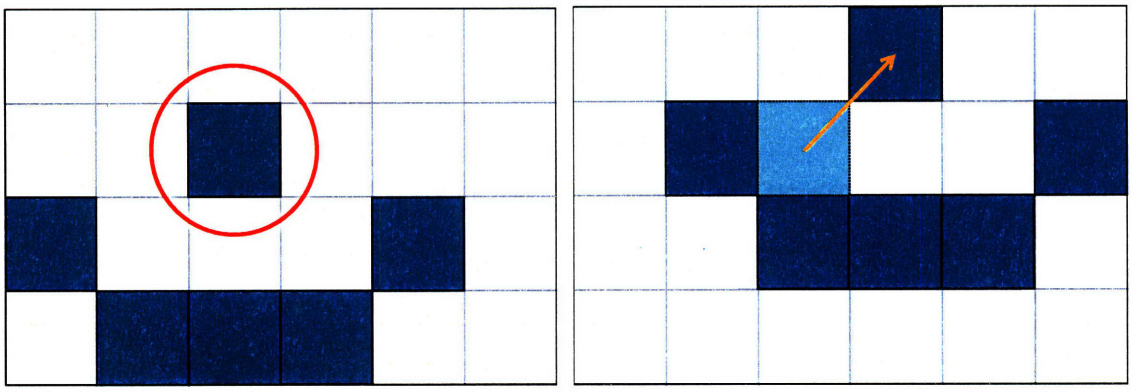
In order to synchronize this flow of images, a simple lockfile scheme is also employed on the ramdisk. When the image puller has finished saving the latest image, it also creates a temporary lockfile. Meanwhile, the image processing thread spins

in a while loop, sleeping 1 ms in between checks of the existence of the lockfile by attempting to delete it. Once the file is successfully deleted, the system knows that the image file is new and loads it into memory and starts an image processing cycle. After finishing processing the image, the thread returns to its wait loop, trying to delete the lockfile until it is again created by the image puller process, and the loop is repeated. By sleeping between deletion attempts, the thread yields it's own process so that other threads may operate in the background and does not consume excessive resources. A sleep time of 1 ms is used to balance the processing latency with the diminishing returns compared with the already large 6 ms penalty simply from decompressing the JPEG.

3.1.2 Feature Tracking

Feature tracking between sequential images is an essential technology for most vision-based motion analysis systems. In a given image, every pixel corresponds to a physical point in three-dimensional space. Using an inverse model of the camera, the corresponding bearing to that point in the camera frame can be explicitly determined. The distance to the point, however, is unknown, unless the same point can be compared between multiple images, either from a second camera, or from a different point in time. If the two images are taken from different locations, then the two bearings will nearly intersect and the approximate location of the point in 3D space can be computed.

The difficulty, however, comes in effectively matching the same point seen between multiple images. This correlation is difficult because there is very little information contained in a single pixel, and the same pixel may not be exactly the same in the next image due to changes in orientation, lighting, or both. The solution adopted in this system is to match surrounding image patches. Image patches are rectangular regions around pixels, typically between 5 and 20 pixels on a side. The patches capture additional contextual information about the pixel and thus can be compared with more certainty than single pixels. For example, Figure 3-1 shows a situation in which there are 6 identical pixels between two images, and we want to track one



(a) Identical pixels form a shape based on relative context. Red circle indicates pixel to track.

(b) Context shows the same shape is repeated, indicating the tracked pixel has moved up and right.

Figure 3-1: Image patch feature tracking allows differentiation between otherwise identical pixels.

of them. The only way to differentiate the pixel of interest is by examining its relationship to the surrounding pixels. In this fashion, it can clearly be seen that the pixel has moved up and to the right, and thus we can now be confident that this new location corresponds to the same point in space that we were trying to track.

In a real system, the correlation between pixels is not as obvious as it is for people to interpret Figure 3-1. The correlations change a little bit over time, stretching and rotating, thus the best match must be found instead of an exact match. The `cvCalcOpticalFlowPyrLK` [30, 45] function does this by breaking down the image into multiple "pyramids" of differing coarseness, used to refine the match successively by differencing the pyramid blocks, increasing computational efficiency. This method is sufficiently accurate to track features to sub-pixel accuracy, and is robust to image deformations from perspective change, image blur, compression artifacts, and other discrepancies. Figures 3-2 and 3-3 show an example of two points on a transmitter being tracked over several frames. Note that, even though the image has rotated and is more blurred in the second image, the features are still tracked. This robustness is due to the incremental nature of the transformations as they are tracked between single frames, which will have minimal differences during normal operation. The effect of the transformations also tend to encompass entire image patches, preserving

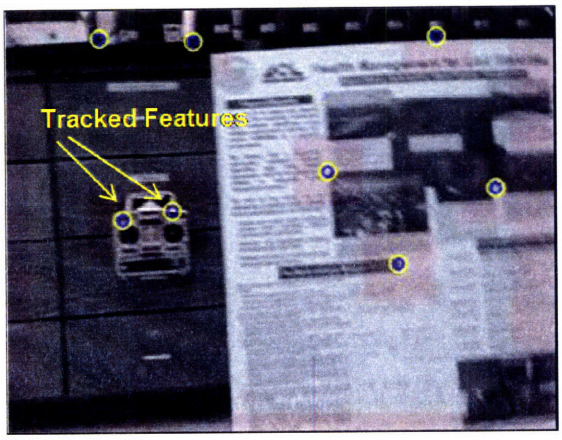


Figure 3-2: A snapshot from the onboard camera capturing features in view.

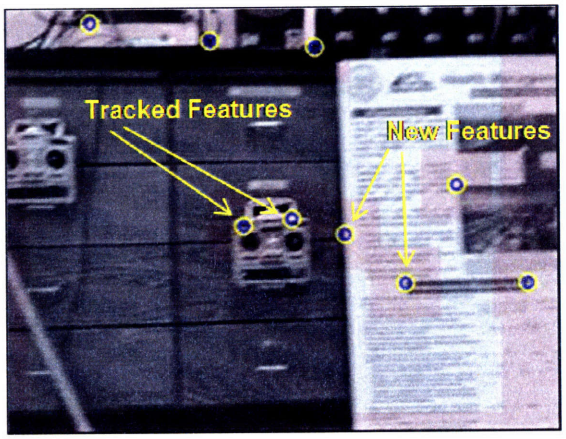


Figure 3-3: The same scene several frames later as the vehicle moves to the left.

the general form of the features so they can still be matched.

3.1.3 Finding and Maintaining Features

Although feature tracking is essential to the image processing, it is also critical that we choose good features for the tracker to track. For our motion estimate, we have very demanding requirements for the tracking, and we have implemented several methods to make and keep the features being tracked as useful as possible. The primary requirement is that they be easily tracked. This is primarily determined by finding points with large eigenvalues of the image gradient in multiple directions, also known as corner detection. The `cvGoodFeaturesToTrack` algorithm also employs some addition measures of feature quality, described in detail by Shi and Tomasi [63].

We also need the features to provide as complete a representation of the environment as possible, thus we want to spread the features out, giving us as much unique information from each feature as possible. The second major reason to spread the features is for environment mapping and obstacle avoidance. Although vision processing systems cannot guarantee they will see all potential hazards (especially not blank walls), there is a significantly higher chance of seeing obstacles if the features are spread over the entire image area. This way, if one object has a lot of texture, all the features will not be dedicated to observing only that one object.

The primary method to maintain feature spread is to choose them far apart from the beginning, which is a supported feature of `cvGoodFeaturesToTrack`. OpenCV also supports the usage of masks which can be placed in the vicinity of existing features, preventing new features from being chosen close to them. We have chosen to use a square masking pattern because, although more intuitive, circular masks require significantly more computation because of the transcendental evaluations required to compute Cartesian pixel coordinates. Finally, OpenCV also includes support for a Rectangle Of Interest (ROI) which will only allow the algorithm to look in a defined sub-region of the image, presumably selected to give the best new feature locations. The masking process has proved to be very computationally intensive, but unfortunately the computational promise of ROI's could not be realized for this thesis because of time limitations.

We have found that our system works very well with a fixed number of features, rather than searching for features in an accumulated batch process. This is because of the real-time nature of the system. Although it uses less computation in total to find several features at once because the good features to track algorithm scans the entire image exactly once for any number of features, saving this time between frames is useless because the time is lost as the system is waiting for camera updates. Thus whenever one or more features are lost, they are replaced immediately.

Although picking features is the obvious point at which to force feature spread, features can become clumped together with time (e.g., during backward movement). In order to maintain feature spread, it is important to remove clumped features and replace them with new ones. Every tracking cycle the features are compared, if two features are within one-half of the minimum distance specified above, the newer feature is marked for removal. Occasionally features also jump because of contextual ambiguities, and these are removed as well.

Feature persistence is also an important element in choosing features. The more measurements there are for a single feature, the better the estimate of its location, so we want to maximize the time that it is in view of the camera. Because the vehicle is moving around, features will often move out of view, but we can predict these future

losses based on present movement of the camera using average translational optical flow, which is simply the average of the differences of all the tracked pixel locations relative to the previous frame, separated into vertical and horizontal components. For example, if the optical flow indicates the camera is moving left, the system will mask out the right edge accordingly so that new features are not selected where they are going to quickly disappear. A good value for this was found to be eight times the average optical flow value, which means features will last at least half a second.

3.1.4 Tracking Implementation

When the vision tracker thread is initialized, the first thing it does is allocate several temporary image buffers and video file writers. The second task is to scan the image for the first set of features using `cvGoodFeaturesToTrack`. A minimum distance if forced that achieves sufficient image coverage, and initializes the system with a set of good features.

After initialization, the vision tracker waits for the next image, during which several filter updates are generated based on the higher frequency IMU data, resulting in estimates of the new feature locations. The algorithm now has a previous feature set to work with and estimates of the new feature locations with which it can attempt to track the persistent features into the new frame efficiently. This is done with a call to `cvCalcOpticalFlowPyrLK`, which generates pyramidal representations of the image for faster matching, a new array of feature locations corresponding to the new image, and a status array indicating features that have been lost, or could not be tracked in the new image.

Once the tracking is done, the tracker scans the remaining features and discards additional features that are clumped together, have jumped, or have poorly converging 3D position estimates. The tracker then examines the complete list of lost features, saves their indices to a `lost_features` array for later reference, and finds replacements, again using `cvGoodFeaturesToTrack`. The difference now, however, is that the image is masked so as to maximize the utility of these new features.

Once the new features are found, they are cross referenced with the `lost_features`

array so they can be slipped into the lost feature positions. The view counts are reset to zero, indicating to the estimation filter they are new features and should be reinitialized, and they have new identification numbers generated so the environment mapper will generate new obstacles in the map. On all subsequent tracking steps, the pyramid generated for the old image can be reused from the previous call to `cvCalcOpticalFlowPyrLK`, saving processor cycles.

3.2 Ego-Motion Estimation

The process for motion estimation has been developed in collaboration with Draper Labs. Draper was previously developing this capability internally using complex simulations based in part on work by Davison, Montiel, and Civera on monocular SLAM [10, 49]. A quick overview of the filter operation is presented below, but is not meant to be a complete presentation of this subsystem.

3.2.1 State Representation and Dynamic Equations

The state is represented in a typical local-level inertial world frame sense, such that gravity is aligned vertically, the world is considered flat, the vehicle moves around in three-dimensional Cartesian space, and the vehicle has an attitude has a Quaternion internal representation (which is transformed to Euler angles and rotation matrices for use outside the state estimator). Equations (3.1) and (3.2) define the vehicle state as

$$\mathbf{X} = \begin{bmatrix} \mathbf{x} & \mathbf{v} & R^{BW} & \mathbf{b}_A & \mathbf{b}_G & \mathbf{y} \end{bmatrix}^T \tag{3.1}$$

$$\mathbf{y}_i = \begin{bmatrix} x_i & y_i & z_i & \theta_i & \phi_i & \rho_i \end{bmatrix}^T \tag{3.2}$$

where $\mathbf{x}$ is the vehicle position, $\mathbf{v}$ is the velocity, R^{BW} is the rotation from body to world frame (which represents current vehicle attitude), and $(\mathbf{b}_A, \mathbf{b}_G)$ are the accelerometer and gyro biases. The total length of the state depends on the number of features used, with sub-states $\mathbf{y}_i$ for each feature i, which are combined to

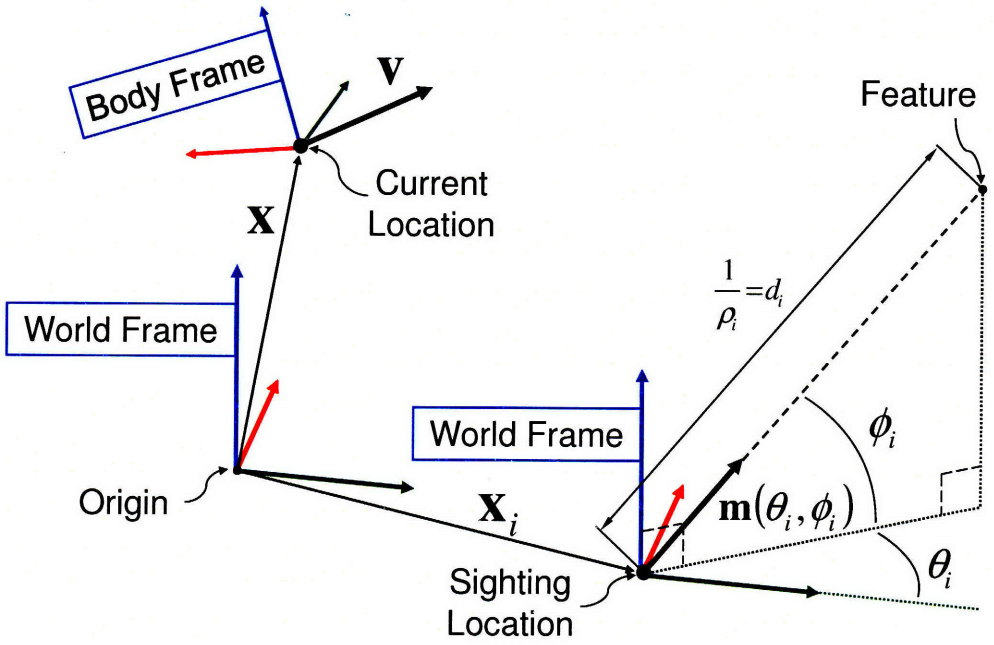


Figure 3-4: Position and velocity are represented in the world frame from the origin. Feature azimuth and elevation bearing angles are measured in the world frame from $\mathbf{x}_i$, the point of first sighting. R^{WB} defines the body frame.

form $\mathbf{y}$. Each feature state $\mathbf{y}_i$ is composed of the vehicle position at first sighting $\mathbf{x}_i = (x_i, y_i, z_i)$, the world frame azimuth and elevation angles (θ_i, ϕ_i) (defined in Figure 3-4), and the inverse of the distance to the feature ρ_i. Ten features were used in our tests as it proved to be a good balance of robustness and computational tractability, with a total state vector length of $5 \times 3 + 10 \times 6 = 75$. When new inertial measurements of specific forces and angular rates $(\mathbf{f}, \mathbf{w})$ are received, the state is

updated according to the following dynamic equations[2]

$$\dot{\mathbf{x}} = \mathbf{v} \tag{3.3}$$

$$\dot{\mathbf{v}} = G + R^{BW}(\mathbf{f} - \mathbf{b}_A) \tag{3.4}$$

$$\dot{\mathbf{R}}^{BW} = [(\mathbf{w} - \mathbf{b}_G)\times] \tag{3.5}$$

$$\dot{\mathbf{b}}_A = -\frac{1}{\tau_A}\mathbf{b}_A \tag{3.6}$$

$$\dot{\mathbf{b}}_G = -\frac{1}{\tau_G}\mathbf{b}_G \tag{3.7}$$

$$\dot{\mathbf{y}} = 0 \tag{3.8}$$

where G is Newtonian gravity and (τ_A, τ_G) are correlation time constants of about 300 seconds for the accelerometers and gyros [46]. These simple updates, referred to as time updates, occur in sync with the inertial measurements, which were at a frequency of 50Hz in our testing. The vision processing is also running independently, but at a much slower rate of about 15 Hz.

3.2.2 Vision Updates

Approximately every three time updates, new vision data becomes available, and a vision update can be performed with an Extended Kalman Filter (EKF). The EKF is designed to efficiently fuse the inertial and vision data using an internal state representation based on estimation error relative to the nominal state, as shown in Equation (3.9)[3].

$$\delta\mathbf{X} = [\delta\mathbf{x} \quad \delta\mathbf{v} \quad \mathbf{\Psi} \quad \delta\mathbf{b}_A \quad \delta\mathbf{b}_G \quad \delta\mathbf{y}]^T \tag{3.9}$$

[2]Note that Equation (3.5) grossly oversimplifies the consideration of the attitude. For insight on this process, see [60].

[3]The use of error dynamics is generally considered more robust than estimating the absolute state directly [46].

where Ψ is the error in body-frame vehicle attitude. The error-state and covariance are propagated by the EKF according to the following dynamics

$$\begin{aligned}
\delta\dot{\mathbf{x}} &= \delta\mathbf{v} && (3.10)\\
\delta\dot{\mathbf{v}} &= R^{BW}\left(\Psi \times \mathbf{f} - \delta\mathbf{b}_A\right) + \omega && (3.11)\\
\dot{\Psi} &= -\left[\mathbf{w}\times\right]\Psi - \delta\mathbf{b}_G + \omega && (3.12)\\
\delta\dot{\mathbf{b}}_A &= -\frac{1}{\tau_A}\delta\mathbf{b}_A + \omega && (3.13)\\
\delta\dot{\mathbf{b}}_G &= -\frac{1}{\tau_G}\delta\mathbf{b}_G + \omega && (3.14)\\
\delta\dot{\mathbf{y}} &= 0 && (3.15)
\end{aligned}$$

where ω is assumed to be Gaussian white noise. The line of site vector $\mathbf{m}(\theta_i, \phi_i)$ is defined in Equation (3.16), and body-frame feature bearings $\mathbf{h}$ are predicted with Equation (3.17)

$$\begin{aligned}
\mathbf{m}(\theta_i, \phi_i) &= \left[\cos(\phi_i)\cos(\theta_i) \quad \cos(\phi_i)\sin(\theta_i) \quad \sin(\phi_i)\right]^T && (3.16)\\
\mathbf{h}_i &= R^{WB}\left[(\mathbf{x}_i - \mathbf{x}) + \frac{1}{\rho_i}\mathbf{m}(\theta_i, \phi_i)\right] && (3.17)
\end{aligned}$$

where $R^{WB} = \left(R^{BW}\right)^T$ is the rotation from world to body frame, and again, $\mathbf{x}$ is the current vehicle location and $\mathbf{x}_i$ was the vehicle location at the first sighting of feature i. These estimated feature bearings are then used to estimate the corresponding pixel locations as projected onto the camera image plane with the inverse pinhole camera model of Figure 3-5 using Equation (3.18)

$$\mathbf{z}_i = \begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} u_0 + f_u\frac{h_{iy}}{h_{ix}} \\ v_0 + f_v\frac{h_{iz}}{h_{ix}} \end{bmatrix} \qquad (3.18)$$

where (f_u, f_v) are the focal lengths in pixels, which are different for the two image axes because the physical pixels of the camera are not square. The filter thus operates in typical predict/update fashion, where $\mathbf{z}$ is used as the prediction to compare against tracked image feature pixel measurements. Ultimately, the measurement matrix and

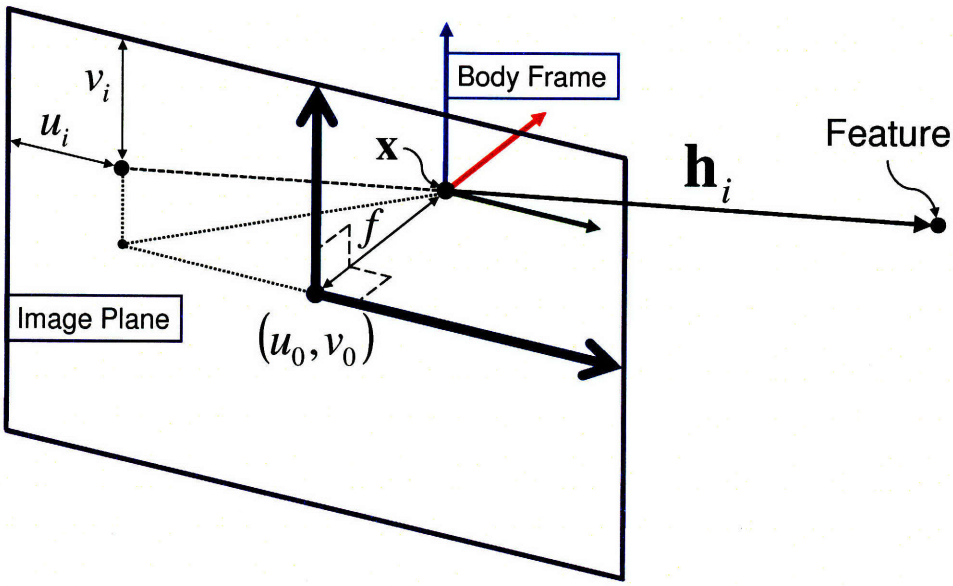


Figure 3-5: Pinhole camera model showing (u, v) pixel coordinates of a tracked feature. The purple box is the image plane, f is the camera focal length, and $\mathbf{h}_i$ is the body-frame vector from the current vehicle position to the tracked feature.

covariance are formed to update the nominal state with the standard discrete EKF update equation (3.19)

$$\mathbf{X}_k = \mathbf{X}_{k-1} + \mathbf{K}_k \left[\mathbf{z}_k - \hat{\mathbf{z}}_k\right] \tag{3.19}$$

with

$$\mathbf{K}_k = \mathbf{P}_{k-1}\mathbf{H}^T \left(\mathbf{H}\mathbf{P}_{k-1}\mathbf{H}^T + \mathbf{R}\right)^{-1} \tag{3.20}$$

$$\mathbf{P}_k = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}\right)\mathbf{P}_{k-1} \tag{3.21}$$

$$\mathbf{H}_i = \frac{\partial \mathbf{z}_i}{\partial \delta \mathbf{X}} = \frac{\partial \mathbf{z}_i}{\partial \mathbf{h}_i}\frac{\partial \mathbf{h}_i}{\partial \delta \mathbf{X}} \tag{3.22}$$

$$\frac{\partial \mathbf{z}_i}{\partial \mathbf{h}_i} = \begin{bmatrix} -f_u h_{iy}/h_{ix}^2 & f_u/h_{ix} & 0 \\ -f_v h_{iz}/h_{ix}^2 & 0 & f_v/h_{iz} \end{bmatrix} \tag{3.23}$$

$$\frac{\partial \mathbf{h}_i}{\partial \delta \mathbf{X}} = \begin{bmatrix} -\rho_i R^{WB} & 0 & [\mathbf{h}_i\times] & \rho_i R^{WB} & \dfrac{\partial \mathbf{h}_i}{\partial \theta_i} & \dfrac{\partial \mathbf{h}_i}{\partial \phi_i} & R^{WB}\left(\mathbf{x}_i - \mathbf{x}\right) \end{bmatrix} \tag{3.24}$$

$$\frac{\partial \mathbf{h}_i}{\partial \theta_i} = R^{WB}\begin{bmatrix} -\cos\left(\phi_i\right)\sin\left(\theta_i\right) & \cos\left(\phi_i\right)\cos\left(\theta_i\right) & 0 \end{bmatrix}^T \tag{3.25}$$

$$\frac{\partial \mathbf{h}_i}{\partial \phi_i} = R^{WB}\begin{bmatrix} -\sin\left(\phi_i\right)\cos\left(\theta_i\right) & -\sin\left(\phi_i\right)\sin\left(\theta_i\right) & \cos\left(\phi_i\right) \end{bmatrix}^T \tag{3.26}$$

where $\mathbf{P}$ is the state covariance, $\mathbf{R}$ is the measurement noise covariance, $\mathbf{z}_k$ is the aggregate of our latest feature tracking measurements, $\hat{\mathbf{z}}_k$ is the nonlinear estimate of the same based on Equation (3.18), and $\mathbf{H}$ is the aggregate of the individual measurement matrix rows $\mathbf{H}_i$. The error-state is zeroed out after every update is applied in order to keep the states near the nominal value about which they were linearized [46]. The updated nominal state estimate is subsequently used by the extended environment mapper and controller, which are discussed further in sections 3.3 and 4.4, respectively.

3.2.3 Ego-Motion Estimation Results

During experimentation, estimating the state was done with two different methods—the first was in an offline context where the vehicle was flown using the high-fidelity truth data from the Vicon camera system for control feedback, resulting in very smooth and stable flights and corresponding imagery. The second method used the vision-based estimate itself to close the loop on the flight control. These methods produced significantly different results because of the coupling between the dynamic motions of the vehicle and the onboard image quality, resulting in degraded estimation performance when using the estimate itself to close the loop.

The RAVEN flight space does not match the description of a target environment as described in section 1.1 because it is purposely cleared for ample flight space and easy viewing of the vehicles. Unfortunately, this arrangement proved too barren for sufficient feature tracking, and additional clutter was added for these experiments. Figures 3-6 and 3-7 show how debris was haphazardly arranged without any particular patterns or geometry, so that no specific structure, such as co-planarity of features, could be assumed. Debris was also specifically arranged to cause complex occlusion, such as from chairs, table legs, and other non-convex obstacles, which is a common weakness of feature tracking in cluttered indoor environments.

In the offline case, the smooth flight from the high-fidelity Vicon tracking provides an ideal data set for estimating ego-motion with vision, even when confronted with a challenging environment. Figure 3-8 shows that over a distance of 3.5 meters and

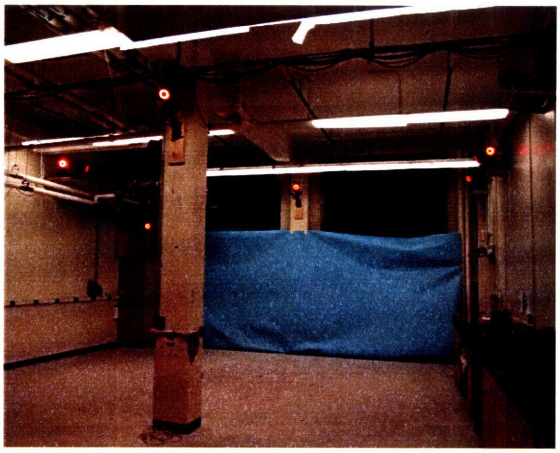


Figure 3-6: Typical RAVEN configuration.

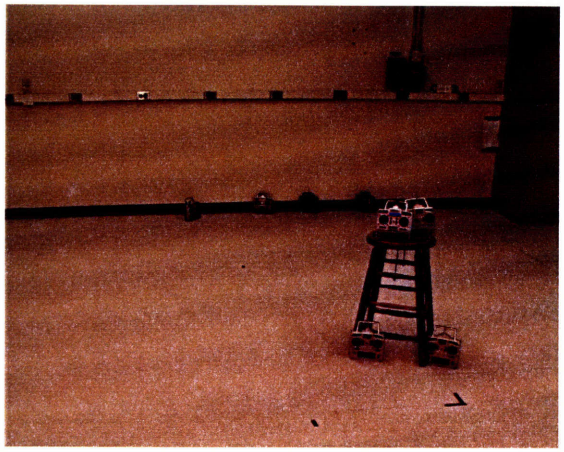
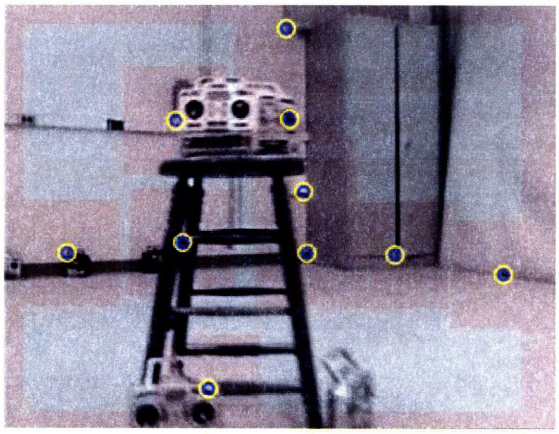


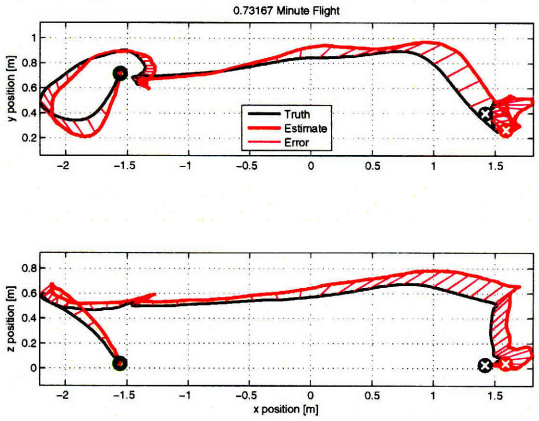
Figure 3-7: RAVEN with added clutter.

a time period of 43 seconds flying around a stool, a maximum of only 28.3 cm of error accumulated, which is about 8.1% at 38.8 cm/min. The primary difficulty with vision in the loop estimation is that estimate errors decrease control performance, generally manifested as oscillations. These oscillations cause the camera view to change constantly, which causes features to go out of view more often, reducing their persistence. The high rate of movement also causes the images to blur because of the low-light indoor conditions and the low camera quality. Because of the decreased vision fidelity, the estimate is degraded further, which causes a positive feedback loop as the control performance and oscillations subsequently decline further.

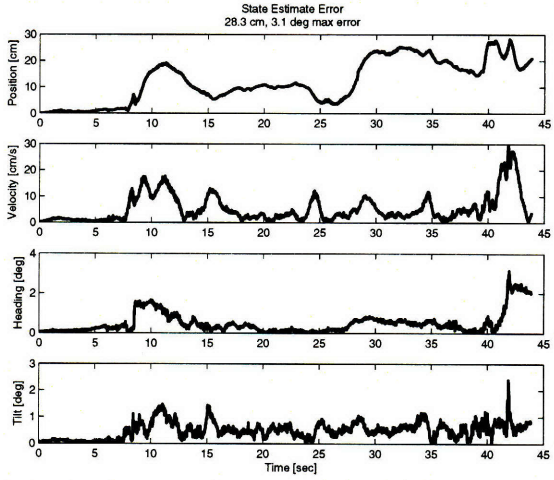
A well designed controller can still fly the vehicle accurately, however. The performance of the system was sufficient for controlled flight inside the small RAVEN environment with no initial knowledge of the environment. Figure 3-9 shows the performance of an example flight, but the maximum error of 64 cm is more than twice as large as the maximum error in the offline case. Approximately the same distance is covered up to the point the maximum is reached, thus the maximum error is about 18.25% at 76.8 cm/min. It is also interesting to note that the error accumulates as the vehicle moves to the right with the estimate lagging behind the truth, then recovers as the vehicle moves back to the left. This is attributed to consistent under-estimate of the vehicle velocity do to a bias in measuring the ranges to features further than



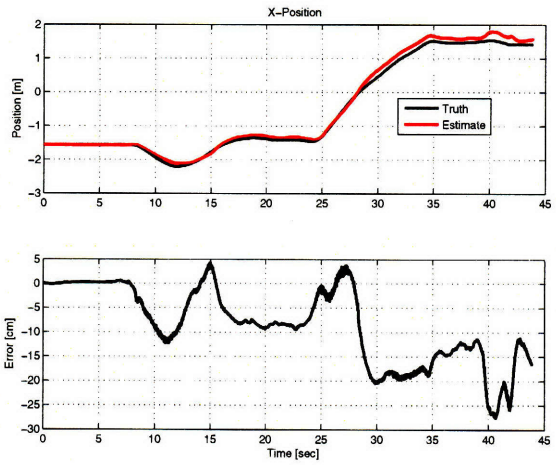
(a) Processed image showing occlusion and blurring.



(b) Orthographic flight path plots. Green circle indicates takeoff, circled X's are touchdown points.

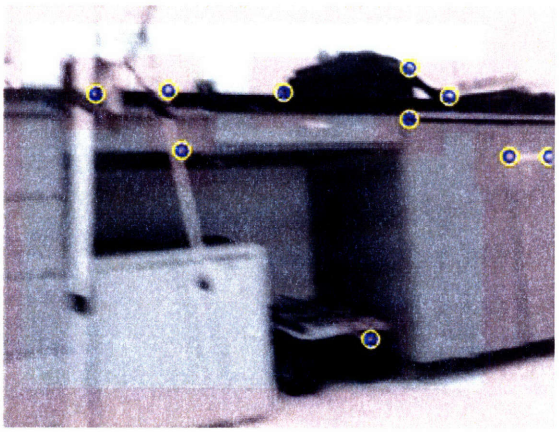


(c) Error propagation of various vehicle states.

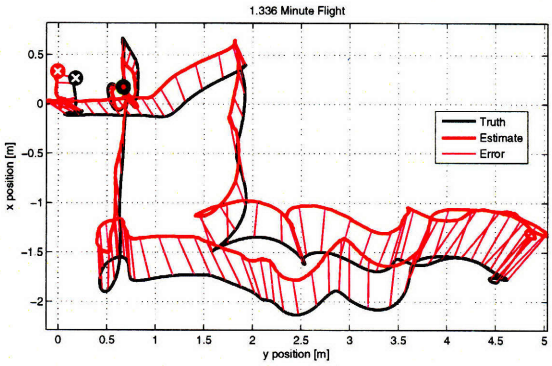


(d) X-axis position estimate error propagation during the flight.

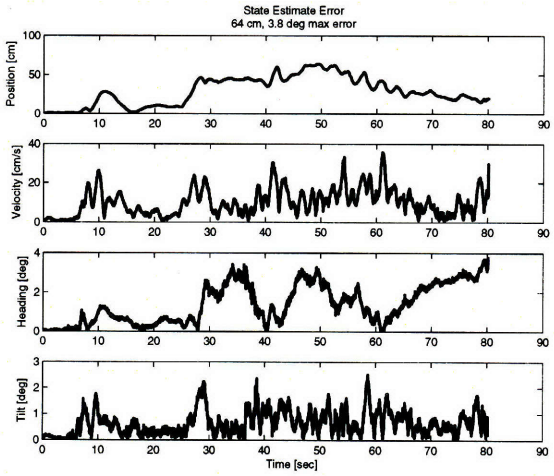
Figure 3-8: Offline ego-motion estimation. Control loop closed with Vicon, waypoints generated by operator.



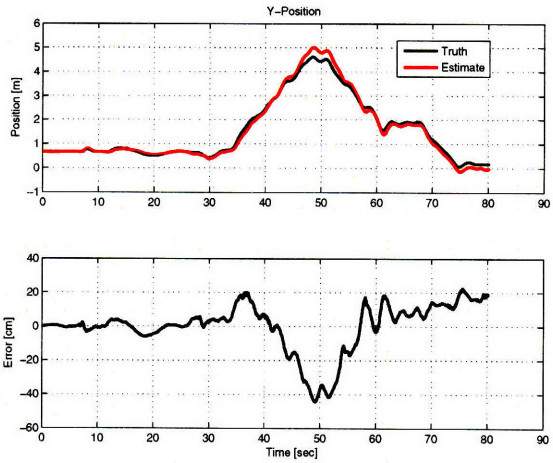
(a) Processed image with heavy image blur due to an aggressive maneuver, presumably from a bad velocity estimate.



(b) Orthographic flight path.



(c) Error propagation of various vehicle states.



(d) Y-axis position estimate error propagation during the flight.

Figure 3-9: Analysis of ego-motion estimate. Control loop closed using the vision-based estimate for feedback, waypoints generated by operator.

they actually are. The exact cause of this bias is not known, but is most likely a product of the fixed feature initialization range.

3.3 Fast Environment Mapping

Estimating and mapping the environment is essential for an autonomous vehicle to navigate in an unknown space. By combining the data from the feature tracking and ego-motion estimation, it is possible to estimate the locations of all the features tracked by the camera, forming a complete three-dimensional map of the environment. This map can be used as information for the operator to reconstruct the remote environment where the vehicle is flying, and as a reference for the vehicle to avoid obstacles autonomously.

There are two methods for estimating feature locations in this system. The first is embedded within the estimation filter, because the feature positions must be estimated in order to estimate the vehicle state. This system is quite computationally expensive, however, and thus only a few features can be tracked at once. The second method employs a simple least-squares pseudo-intersection algorithm which is very fast, and thus can track many more features and give a much more complete view of the world.

The concept behind this fast mapping is that every measurement of a feature can be interpreted as a ray in three-dimensional space—the pinhole camera model from Equation (3.18) converts the pixel locations into relative bearing vectors in the body frame. The estimated attitude of the vehicle is used to rotate the bearings into the world frame, and the estimated position of the vehicle is combined with the world-bearings to define three-dimensional world-frame rays for each feature. As the vehicle moves through space, each new image can be processed to determine a new set of rays. Because the features are tracked between frames, the new rays can be correlated to the previous rays, forming separate sets of rays for each feature. Because the tracked features correspond to physical points in 3D-space, every feature is assumed to be an

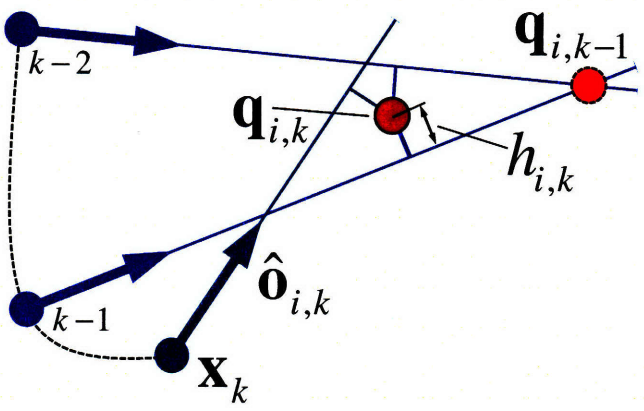


Figure 3-10: Vision bearings as the vehicle traverses dashed path.

obstacle[4], and each set of rays will have a location where all the rays are most near intersection.

3.3.1 Pseudo-Intersection Optimization

Figure 3-10 shows the formulation for optimally estimating the psuedo-intersection point $\mathbf{q}_{i,k}$ of obstacle i at time k. $\mathbf{x}_k$ is the estimated vehicle position, $\hat{\mathbf{o}}_{i,k}$ is the corresponding unit vector bearing estimate from the vehicle to feature i, $h_{i,k}$ is the minimum distance from $\mathbf{q}_{i,k}$ to the estimated vision ray, and two previous bearing estimates at times $k-1$ and $k-2$ show the approximate nature of the intersection. In general, $\hat{\cdot}$ denotes unit or normalized vectors, such that $\hat{\mathbf{g}} = \mathbf{g}/\|\mathbf{g}\|$. A simple least-squares algorithm efficiently computes the best-fit intersection point $\mathbf{q}_{i,k}$ by minimizing $\sum h^2$. This pseudo-intersection optimization, derived by Bethke in [3], has a very unique structure that makes it very computationally efficient for aggregating measurements over time, as we are doing here. Given a set of n estimated ray bearings $\hat{\mathbf{o}}$, and n corresponding origins $\mathbf{x}$, we compute the optimal three-dimensional position estimate of a feature $\mathbf{q}$, with the following equation

[4]This is a valid assumption in most scenarios, although smoke and other complexities could be perceived without posing a real threat

$$\mathbf{q} = \mathcal{A}^{-1}\mathbf{b} \tag{3.27}$$

where

$$\mathcal{A} = \sum_{i=1}^{n} w_i \left(I - \hat{\mathbf{o}}_i \otimes \hat{\mathbf{o}}_i\right) \tag{3.28}$$

$$\mathbf{b} = \sum_{i=1}^{n} w_i \left(\mathbf{x}_i - (\mathbf{x}_i \cdot \hat{\mathbf{o}}_i) \cdot \hat{\mathbf{o}}_i\right) \tag{3.29}$$

where $\mathbf{a} \otimes \mathbf{b} = \mathbf{a}\mathbf{b}^T$ is an outer product. This is done for each feature being tracked, forming a point-cloud representation of the environment suitable for obstacle avoidance. The original formulation was developed for simultaneous measurements from multiple vehicles tracking the same target, but in the single camera case, each measurement i is actually taken at a different point in time, thus the summations of $\mathcal{A}$ and $\mathbf{b}$ from Equations 3.28 and (3.29)can be accumulated over time for each obstacle as each new bearing is measured. Because our position estimate tends to drift, we also add an exponential decay to slowly attenuate the contributions of the older measurements according to the α parameter in Equations (3.31) and (3.32). This causes the system to weight the more recent feature measurements more heavily, accounting for drift over time[5]. $\mathcal{A}_0$ and $\mathbf{b}_0$ are initialized to zero, and for every new measurement, a simple update produces a new estimate at current time k, which is recomputed for each feature that has a new measurement this timestep

$$\mathbf{q}_k = \mathcal{A}_k^{-1}\mathbf{b}_k \tag{3.30}$$

where

$$\mathcal{A}_k = \alpha\mathcal{A}_{k-1} + \left(I - \hat{\mathbf{o}}_k \otimes \hat{\mathbf{o}}_k\right) \tag{3.31}$$

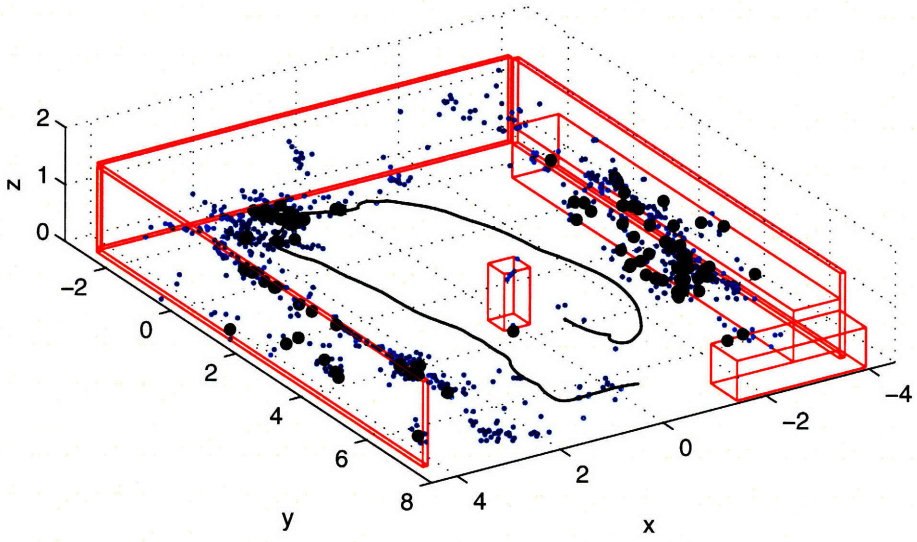
$$\mathbf{b}_k = \alpha\mathbf{b}_{k-1} + \left(\mathbf{x}_k - (\mathbf{x}_k \cdot \hat{\mathbf{o}}_k) \cdot \hat{\mathbf{o}}_k\right) \tag{3.32}$$

[5]This feature turns out to only be useful in cases of persistent observation because the rapid feature turn over while traversing an environment means features are not viewed long enough for their estimates to be effected by drift.

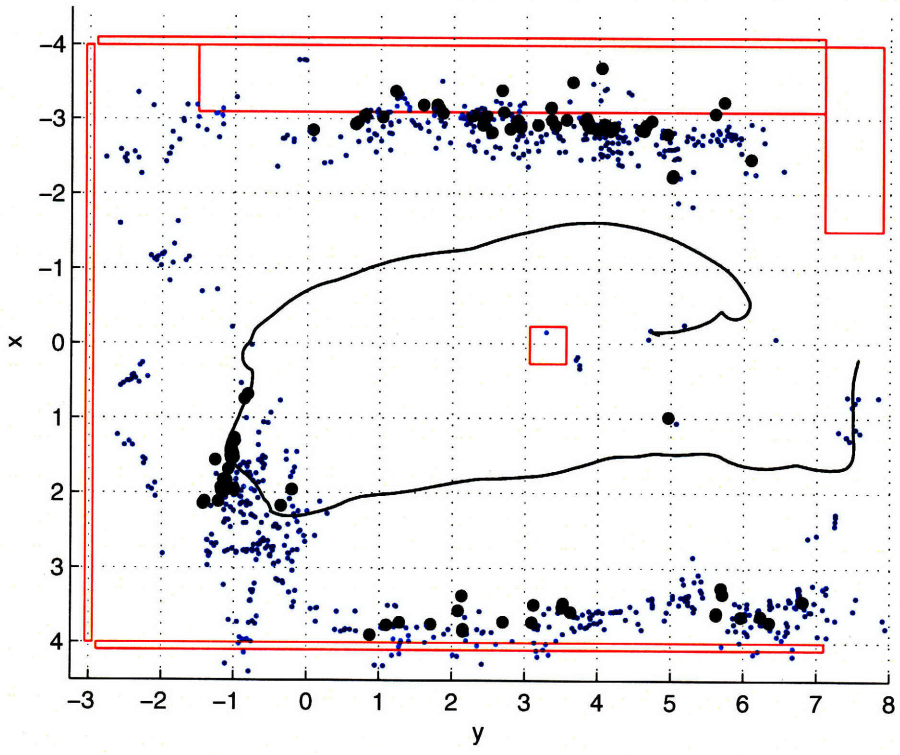
Thus resource and computation requirements are minimal, as we only need to store the 3x3 $\mathcal{A}$ matrix and 3-element $\mathbf{b}$ vector for each feature, and update them with a constant-time expression, independent of the number of measurements we have accumulated over time. The time to update the entire environment based on any number of measurements is thus proportional to the size of the map, which can be limited to define an upper-bound on processing time for mapping. The result is a cloud of physical points in 3-space, much like what would be generated by a LIDAR system, although with significantly lower range estimation accuracy, and lag associated with need to aggregate measurements over time. Figure 3-11 shows an example of how the world can look like a LIDAR point cloud with quite accurate mapping capabilities. The blue dots are from the binocular mapping research developed by He in [24], and the larger black dots are based on the monocular fast mapping algorithm presented in this section. No attempt was made to remove worse features, and external Vicon tracking was used for localization in both cases. The similarity in the data shows that motion parallax can operate as as effectively and accurately as binocular parallax from two cameras. In addition to being configured to track more features, the binocular can estimate depth immediately, thus features that appear for only a short time before going out of view can be accurately mapped, which is why there are more blue points. The obvious advantage of the binocular vision not shown here is that it can also map and track dynamic obstacles, but this did not factor into our testing as the RAVEN room is static.

3.3.2 Map Estimation Confidence

This method on its own, however, does not provide a usable map of the environment because there are many extraneous measurements caused by incorrect feature tracking, primarily from specularity and occlusion. For this reason, it is essential to be able to generate a good confidence estimate for each of the features so we can remove bad estimates (which are probably not physically meaningful), and only consider the best ones. Equation (3.33) shows how this is done by dividing the sum of the history of camera parallax magnitudes $\|\mathbf{\Delta x}\|$ by the most recent position estimate jitter. Sum-



(a) Isometric view showing clouds of feature points.



(b) Orthographic top view.

Figure 3-11: Comparison of binocular (blue) and monocular (black) mapping techniques using identical video and camera localization data. Red lines indicate hard-coded environment features for reference. Binocular mapping and comparison plots provided by Ray He [24].

ming the parallax is a simple and effective way of judging how many different angles the feature has been seen from which helps triangulate the feature more accurately, and the jitter is a direct way of relating to the uncertainty of a changing estimate, like a simplified covariance heuristic, although the ultimate value has no physical meaning—The darkness and length of the feature ellipses in the 3D visualization are based on the confidence such that they look good. The estimate jitter $\|\Delta\mathbf{q}\|$ is simply the magnitude of the difference between the current and previous estimates $\mathbf{q}$.

$$c = \frac{\sum_{i=1}^{n} \|\Delta\mathbf{x}_i\|}{\|\Delta\mathbf{q}\|} \tag{3.33}$$

As with the estimate of obstacle position from Equation (3.28), the confidence can be built incrementally as well. Every time a feature is updated, the distance the vehicle moved since the last update, computed as $\|\mathbf{v}dt\|$, is added to a confidence sum $\tilde{c}_k$. This sum is stored as a single number for each feature, in addition to $\mathcal{A}$ and $\mathbf{b}$. The final confidence at time k is determined by dividing this sum by the distance between the current and previous feature position estimates

$$\tilde{c}_k = \tilde{c}_{k-1} + \|\mathbf{v}dt\| \tag{3.34}$$

$$c_k = \frac{\tilde{c}_k}{\|\mathbf{q}_k - \mathbf{q}_{k-1}\|} \tag{3.35}$$

Once these confidences are determined, they are used to both define when a feature is used for obstacle avoidance, and when a feature should be culled from the map completely. For culling, features are given a grace period of 1 second in which the confidence is expected to develop, after which any feature with a confidence below the specified threshold is removed from the feature list. Without this grace period, features are never successfully formed and the map will not develop. When processing obstacle avoidance, features are only considered if they are above a different, higher threshold, thus newly formed features too young to be culled are not considered. This prevents the vehicle from overreacting to glitches and inaccurately mapped features. Thresholds were tested empirically to obtain suitable behavior.

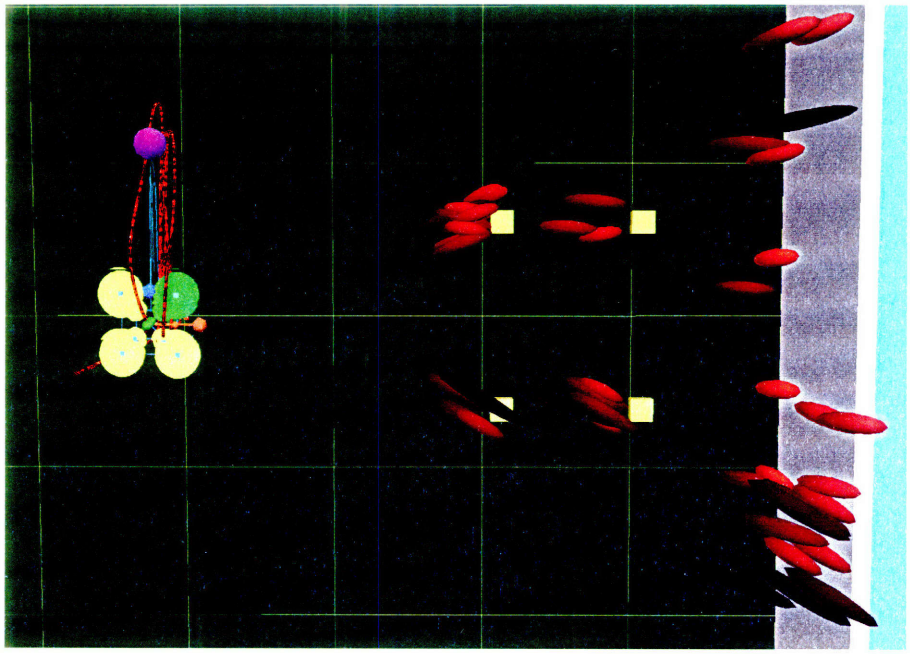


Figure 3-12: Mapping simple, known features gives an indication of the mapping accuracy.

Figure 3-12 shows an example of the interpretation of these confidences in mapping a known array of objects. Features are represented as red ellipsoids which shorten in length and brighten in color as their confidence increases, and always have their long axis aligned in the direction of last sight because the depth the the least well known parameter in the obstacle position estimate. The vehicle was commanded under Vicon control to simply fly back and forth, maximizing the visual parallax to best estimate the feature locations. It is clear the block on the top left is the most easily seen, as it has the most features of the smallest size. The bottom block is near the edge of the field of view of the camera, and the two right blocks are partially occluded by the front blocks. The thin yellow lines on the green ground form 1 meter squares which can be used to approximately judge the 1.5 m parallax movement of the vehicle, and about 0.3 m error in any given obstacle.

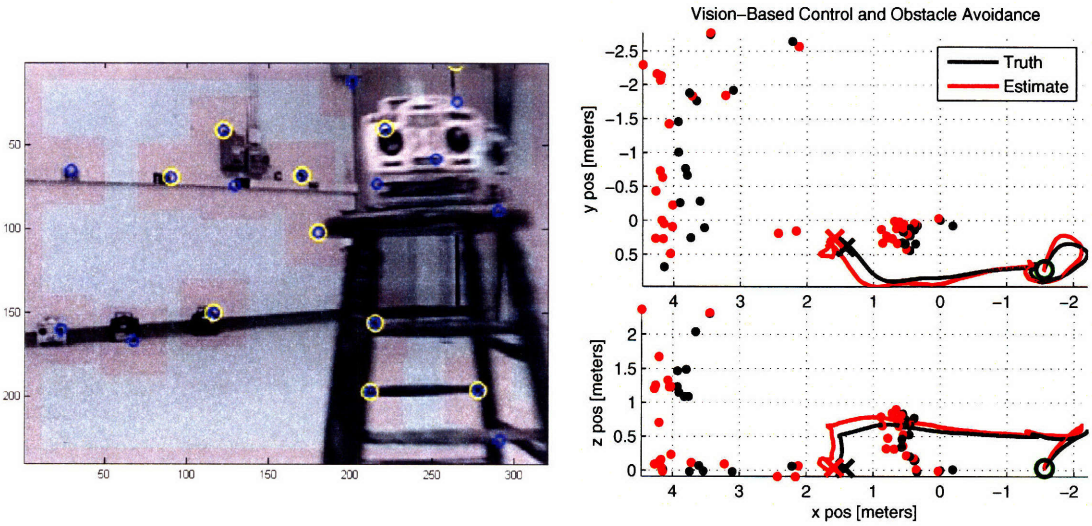
When the map grows beyond a specified maximum size, older obstacles are removed until the map is reduced down to its upper limit, in essence erasing the vehicle's memory. A map size of 100 obstacles was found to be a good balance between the vehicle's need to be aware of nearly obstacles that have moved out of view, and storage

limitations. Because the vehicle tends to look forward as it flies, potential obstacles have a high likelihood of being in view. Old features also lose accuracy over time as the vehicle position estimate drifts, thus this memory loss has proved not to be much of an issue. Additionally, the offboard planner has far fewer storage restrictions and can maintain a much larger map of the environment for more advanced planning over greater distances, potentially building abstract node-graphs of regional connectivity.

Figure 3-13(a) shows an image from the onboard camera system while performing a simple mapping task of a stool and a wall in a relatively sparse environment. Figure 3-13(b) shows a comparison in the mapping performance when using the vision-based ego-motion estimate vs. the Vicon truth to transform the body-frame bearings into the world frame. Note the "truth" features are not actually known because of inaccuracies in the feature tracking, but offline comparisons with physically measured feature locations indicate they are quite accurate. Although the absolute accuracy of the mapping estimate is good, it is not very important. The primary figure of merit for obstacle avoidance is the accuracy of mapping relative to the vehicle position estimate itself.

3.3.3 Complimentary Offsets in Mapping and Localization

In Figure 3-14, the vehicle (orange squares) needs to avoid the obstacle in front of it (red blobs), but the vehicle has accumulated a position estimate error 1 meter to the right (dashed outline) during several minutes of flight. Although this is a large error, the map is also generated based on this incorrect estimate, using the two estimated bearings (dashed arrows from two different points in time), and thus the estimated feature position is also shifted to the right, indicated by the dash-outlined red ellipse. The vehicle plans a path around the estimated object location (blue line) which it thinks it follows accurately (black dashed line). Even though the planned path goes right through the real object (small red circle), the vehicle is actually safe because everything in the physical world is shifted to the left—essentially the position estimate error and mapping error have canceled out, and the vehicle is capable of flying safely indefinitely, with an arbitrary amount of accumulated drift. These canceling errors



(a) The circles indicate points being tracked—yellow for ego-motion estimate filter, blue for mapping.

(b) Comparison in mapping using Vicon truth vs. Vision/IMU estimate for vehicle localization.

Figure 3-13: Environment mapping.

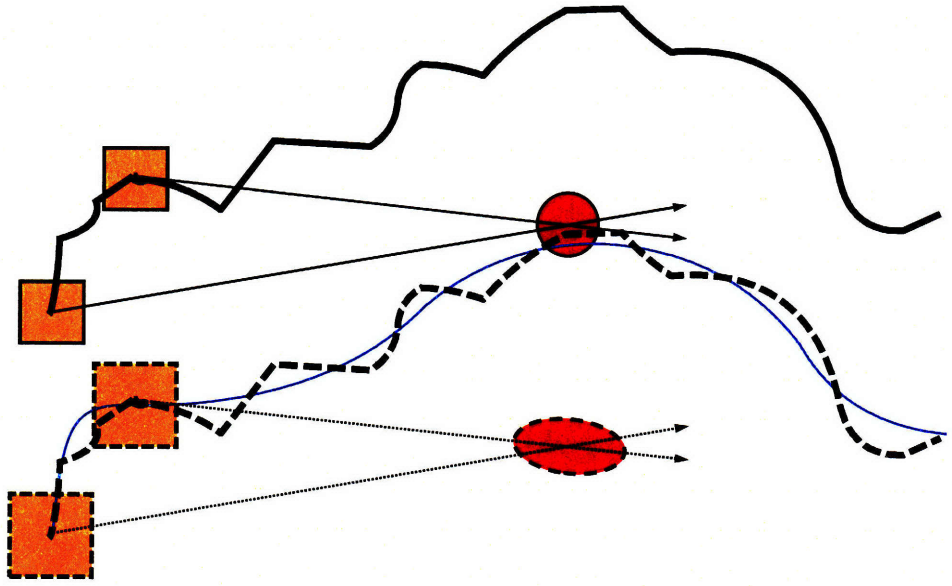


Figure 3-14: The red obstacle only needs to be mapped in a relative sense for the vehicle to avoid it.

are referred to as complimentary offset errors, and are an important attribute to our mapping as they are essential to reliable obstacle avoidance during extended flights. There is still error in the relative estimate from recent position estimate errors, but in the flight test data presented in Figure 3-13, the range to the features shows errors of about 30 cm over 4 meters, or 7.5%, which is very close to the ego-motion estimate error described in section 3.2. This error also tends to be one of scale, thus if the vehicle overestimates the range to a feature, it's probably linked to an overestimate of its own velocity, and thus the two scale errors will cancel out as well, supporting that the error is due primarily to the vehicle state estimate error, and the mapping is quite accurate in a relative sense.

Chapter 4

Guidance and Control

Although an estimate of the state of the vehicle using only visual and inertial data is challenging, guidance and control are also essential for the successful flight of a HMAV. The basic control approach is relatively simple because the pitch and roll tilt axes are already stabilized at 1 kHz onboard the vehicle. The control signals sent to the vehicle through the R/C link are absolute pitch and roll angles, yaw rate, and collective throttle [2]. Neglecting higher order effects, Quadrotors behave like simple integrators with horizontal accelerations directly related to the tilt angles [25, 69]. The loops for each axis are closed with feedback on position, velocity, and heading errors, plus position error integration, with gains heavily weighted toward velocity damping and position error integration to smooth vehicle motion. Tracking accuracy and overshoot are of less concern because the position and mapping estimates themselves are not perfectly accurate, thus disturbance rejection and smooth motion are favored over tracking performance.

The vehicle must be able to take off and land vertically, as well as hover and follow waypoints. Good performance in these various regimes is achieved with a state-machine switching controller. Takeoff is the most challenging because the vehicle needs to fly up out of the ground effect with an unknown battery charge. The first mode of the state machine is standby, during which the vehicle is completely deactivated and the rotors do not spin. Upon receiving a takeoff command, the vehicle starts the rotors spinning and enters wind-up mode. The collective command is

increased at a fixed rate from zero until the vehicle altitude increases 5 cm, at which point the vehicle switches to takeoff mode and the momentum of the vehicle will carry it through the ground effect smoothly. In takeoff mode, the position integrator is doubled relative to the nominal value in order to quickly correct for imbalances in the vehicle. As soon as the vehicle is within 10 cm of the reference position, it is considered trimmed and switches into flying mode where all control gains are set to nominal values. Hovering, waypoint following, and landing have shown sufficient dynamic similarities that a good trajectory will let a single controller fly the vehicle in all conditions. Landing accuracy is not excellent because of ground effect concerns, but it is unclear how to compensate for this without destabilizing the state estimate with over-aggressive gains.

4.1 Vehicle Dynamics

Although the vehicle accepts tilt angles for control input, there is a physical limitation on the speed with which it can achieve the desired attitude. Besides the lag associated with the electronics and software, the vehicle needs to change the torque to the motors, causing a rotational accelerate of the propellers which changes their speed. As the propeller speeds change, an increased lift differential forms which generates a torque about the desired axis of the vehicle. This torque causes an angular acceleration, causing the vehicle to rotate about that same axis. Half way to the target angle the process needs to be run in reverse such that the vehicle decelerated in time to a stop at the desired angle. Figure 4-1 shows this process happening in flight. There is noticeable lag between the commanded tilt angle and the physical angle of the vehicle, primarily due to the

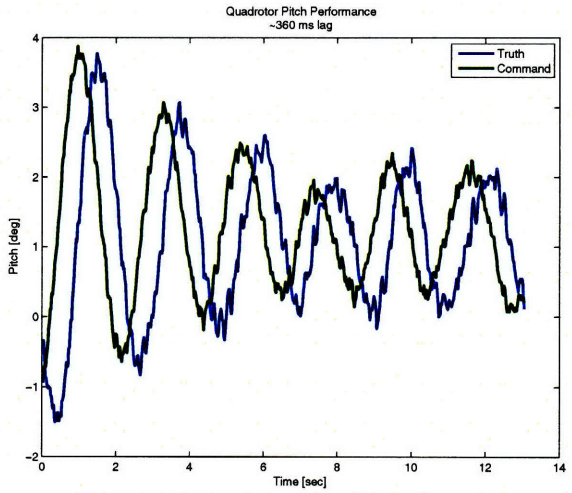


Figure 4-1: Pitch command tracking.

dynamics mentioned above, measured to be about 360 milliseconds.

4.2 Trajectory Following

Accurate trajectory following is important for well controlled flight in constrained environments, and this system demonstrates performance sufficient to avoid mapped obstacles safely, presented in Chapter 5. The primary task for achieving accurate trajectory following is the generation of dynamically appropriate, natively three-dimensional trajectories. A finite acceleration model is used which strikes a balance between good performance and computational simplicity. This method is also very flexible and allows smooth integration with the natively 3D obstacle avoidance implementation described in section 4.3.

The system is fundamentally based on calculating the desired acceleration, which is composed of goal attraction, collision avoidance, and orbital cancellation components. Once the desired 3D acceleration at time k is determined ($\mathbf{a}_k$), it is used to integrate the reference states according to Equations (4.2) and (4.3) to produce a reference velocity and position $(\mathbf{x}, \mathbf{v})$ which are used for controller feedback. If the vehicle is in motion when a new waypoint is received, the previous reference velocity and position values are carried over to propagate the trajectory smoothly and continuously.

$$\mathbf{a}_k = \mathbf{a}_g + \mathbf{a}_c + \mathbf{a}_r \tag{4.1}$$

$$\mathbf{v}_k = \mathbf{v}_{k-1} + \mathbf{a}_k dt \tag{4.2}$$

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{v}_k dt \tag{4.3}$$

The fundamental acceleration component that gets the vehicle to the goal is the goal attraction acceleration $\mathbf{a}_g$, defined in Equation (4.4). When the vehicle accepts a new waypoint, it is commanded to accelerate toward it according to $\mathbf{a}_g$, the magnitude of which is a simple function of the desired cruise speed of the waypoint s_c such that the time to accelerate to the cruise speed is always τ_g, which for our tests was 1 second.

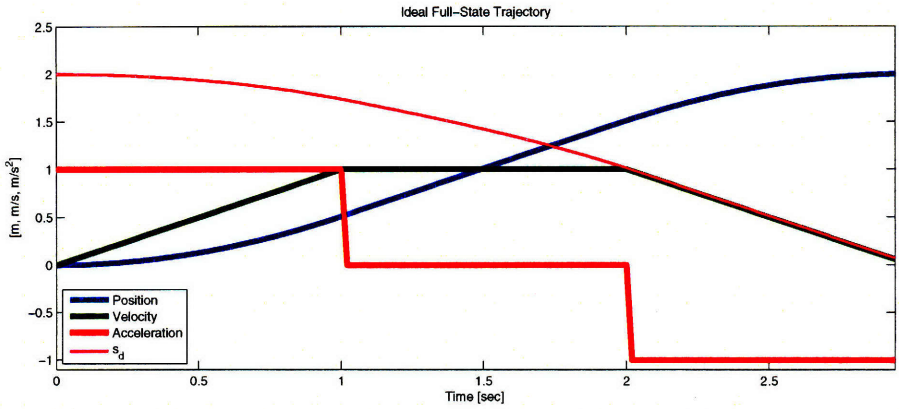


Figure 4-2: An ideal movement trajectory simulated in one dimension.

Once the position reference reaches the cruise speed, $\mathbf{a}_g$ is set to zero, and the velocity reference stays fixed while the position reference integrates forward at constant rate. In this cruise mode (and the acceleration mode), the controller is constantly checking the distance to the goal $\|\mathbf{g}\|$ to determine when to slow down. Equation (4.6) is used to compute s_d, the speed at which the vehicle should be traveling in order to arrive at the goal with zero velocity if the vehicle were to decelerate continuously according to $\mathbf{a}_g$. As soon as the current speed in the direction of the goal s_g is greater than or equal to s_d, $\mathbf{a}_g$ is set to $-a_g\widehat{\mathbf{g}}$ and the velocity ramps down to zero as the position reference comes to rest precisely at the goal point. In summary,

$$\mathbf{a}_g = f(s_g, s_d, s_c)\, a_g \widehat{\mathbf{g}} \tag{4.4}$$

$$a_g = s_c/\tau_g \tag{4.5}$$

$$s_d = \sqrt{2a_g\|\mathbf{g}\|} \tag{4.6}$$

$$s_g = \mathbf{v}\cdot\widehat{\mathbf{g}} \tag{4.7}$$

$$f(s_g, s_d, s_c) = \begin{cases} -1 & \text{if } s_g \geq s_d \\ 1 & \text{if } s_g < s_c \text{ and } s_g < s_d \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

where $\mathbf{g}$ is the vector from the vehicle to the goal. Figure 4-2 shows an ideal trajectory of this type.

Equations (4.9)–(4.12) shows a derivation of s_d based on simple kinematics. The time t_d necessary to decelerate from s_d to 0 with constant deceleration $-a_g$, and the distance x_d traveled during this time, are based on standard Newtonian kinematics. We solve for the speed s_d by setting $x_d = \|g\|$ such that the vehicle speed would reduce to zero over the distance to the goal if it started at speed s_d.

$$t_d = \frac{0 - s_d}{-a_g} = \frac{s_d}{a_g} \tag{4.9}$$

$$\begin{aligned} \|g\| = x_d &= -\frac{1}{2} a_g t_d^2 + s_d t_d \\ &= -\frac{1}{2} a_g \left(\frac{s_d}{a_g}\right)^2 + s_d \frac{s_d}{a_g} \\ &= -\frac{1}{2} \frac{s_d^2}{a_g} + \frac{s_d^2}{a_g} \end{aligned} \tag{4.10}$$

$$\|g\| = \frac{1}{2} \frac{s_d^2}{a_g} \tag{4.11}$$

$$\Rightarrow s_d = \sqrt{2 a_g \|\mathbf{g}\|} \tag{4.12}$$

4.2.1 Orbital Cancellation

The system works very well in this simple form when waypoints are taken one at a time such that the vehicle pauses between each, but if a new waypoint is given while the vehicle is in motion toward another one, an acceleration directly toward the new waypoint, as previously described in Equation (4.4), will result in an orbit resembling planetary motion in which the vehicle never reaches the goal. This is countered by an orbital cancellation term, which adds an acceleration negative to cross-goal velocity component $\mathbf{v}_\perp$ divided by τ_c (which in our tests was 0.5 s), according to Equation (4.14) and Figure 4-3.

$$\mathbf{v}_\perp = \mathbf{v} - \hat{\mathbf{g}} \cdot (\mathbf{v} \cdot \hat{\mathbf{g}}) \tag{4.13}$$

$$\mathbf{a}_c = -\mathbf{v}_\perp / \tau_c \tag{4.14}$$

The construction of $\mathbf{v}_\perp$ is based on simple 3D geometry. The $\mathbf{v} \cdot \hat{\mathbf{g}}$ term gives the length

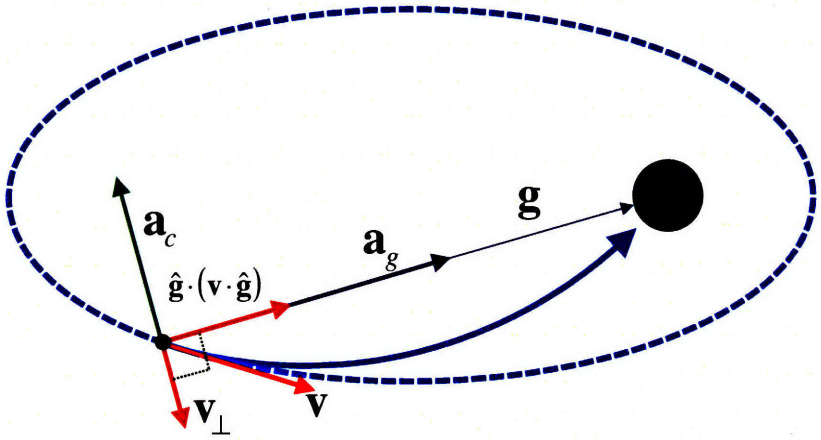
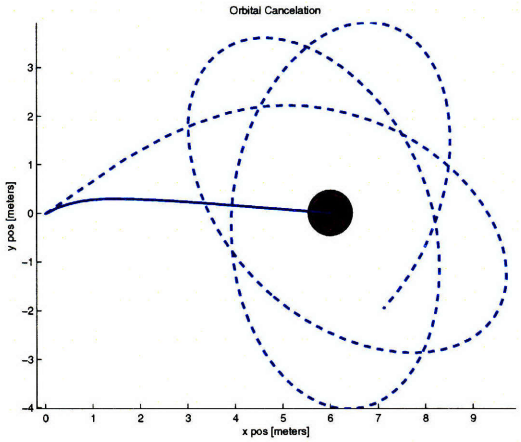
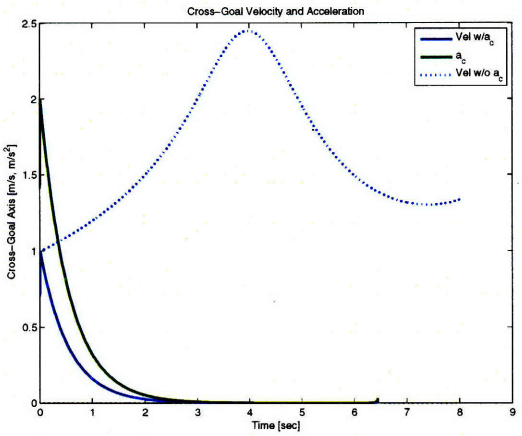


Figure 4-3: Acceleration components toward the goal and perpendicular to the goal vector combine to fly the vehicle directly into the goal.



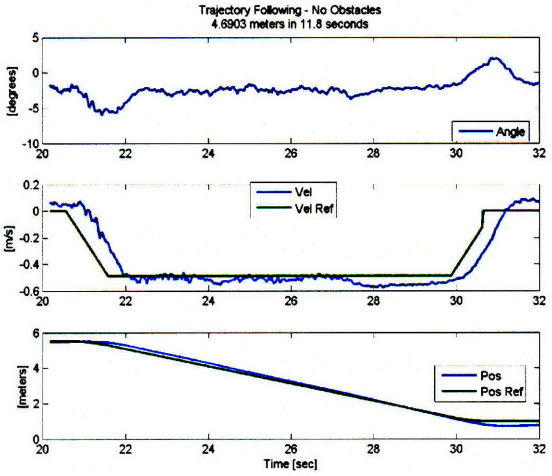
(a) Dashed line has ($\tau_c = \infty$), solid has τ_c = 0.5 seconds.



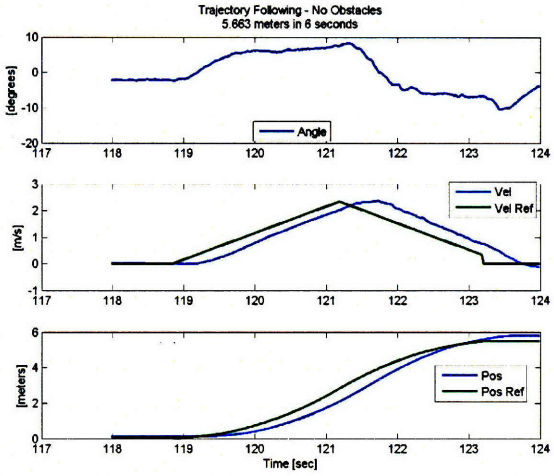
(b) Cross-goal velocity and acceleration magnitudes used to cancel out orbital effect.

Figure 4-4: Simulation of trajectory generation with initial cross-goal velocity. Orbits are non-celestial because $\mathbf{a}_g$ is not a function or r^2.

of the projection of $\mathbf{v}$ onto $\hat{\mathbf{g}}$, which is vectorized by multiplication with $\hat{\mathbf{g}}$ as shown in Figure 4-3. Subtracting this vector projection from the original velocity vector gives the perpendicular component that is in the plane of the two original vectors. This method of deriving perpendicular components is used in several instances of this thesis, particularly for obstacle avoidance in section 4.3. Figure 4-4 shows a simulated comparison of the trajectory generation with orbital cancellation and without.

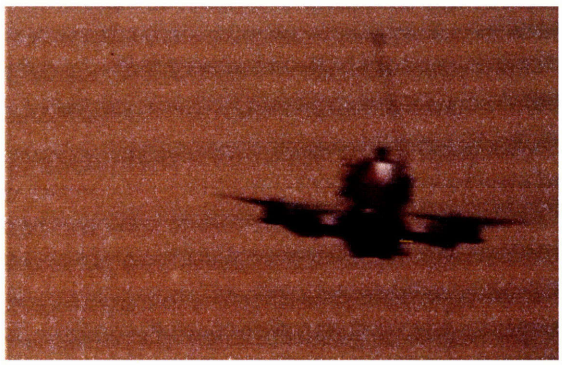


(a) Slow trajectory.



(b) Aggressive trajectory.

(c) Coming to a stop.

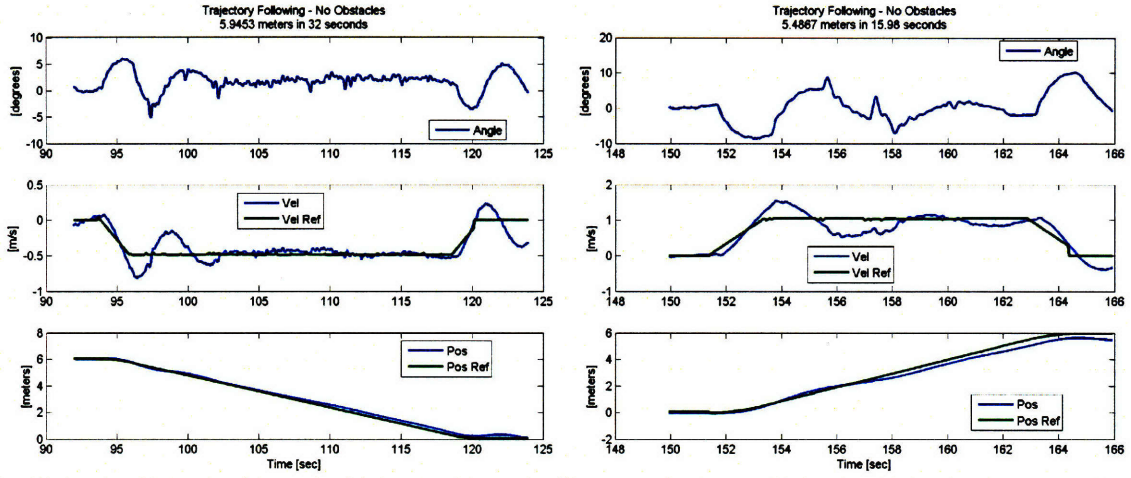


(d) Starting to slow down.

Figure 4-5: Trajectory following.

4.2.2 Trajectory Following Flight Tests

In Figure 4-5(a), the performance while tracking a slow movement plan is presented. Because of the low speed, disturbances have a proportionately larger effect on the system, and aerodynamic damping is negligible. As a result, oscillations are more pronounced in the velocity. The more aggressive maneuver in Figure 4-5(b) shows how the linear velocity profile is similar to the natural dynamic path for the quadrotor, minimizing the appearance of oscillations. The high speed trajectory also demonstrates the initial velocity overshoot, which is a product of the lag in the vehicle dynamics as it tilts over to begin accelerating. The deceleration period is also well matched to the trajectory, almost completely eliminating position overshoot, although there is still a small overshoot from the lag in the vehicle dynamics. Figures 4-5(c)



(a) Slow trajectory.

(b) Aggressive trajectory.

Figure 4-6: Trajectory following with angle reference.

and 4-5(d) show the vehicle coming to a stop. Notice in Figure 4-5(d) the vehicle is both tilted and blurred due to the simultaneous high acceleration and velocity. Figure 4-5(c) is tilted for peak acceleration, but is not blurred because the vehicle has almost finished coming to a stop. Note step in velocity at the end of deceleration. These are due to numerical integration errors and complications from orbital cancellation terms that prevent the real solution to work as well as the simulation, but have shown to have minimal effect in flight performance.

It is also of interest that attempting to remove this acceleration lag is of detriment to the performance of the vision tracking. Adding an attitude reference for feedback increases the dynamic performance significantly by reducing the acceleration lag, visible in Figure 4-6, but does so with much higher angular rates, causing the imagery to blur, reducing the estimation performance as discussed in section 3.2. The solution would be to further smooth the trajectory in order to eliminate steps in the acceleration, such as with a constant-jerk model, but this was deemed unnecessarily complex relative to the performance improvement.

4.3 Obstacle Avoidance

We have developed a lightweight reactive obstacle avoidance heuristic which shows promising results for high-speed, effectively, natively three-dimensional navigation in complex environments, which is easily tractable onboard existing micro flight controllers. It is based on simple closed-form expressions to compute cross-goal accelerations using point cloud map representations, which fits in directly with the full-state trajectory generation we have implemented based on the integration of finite accelerations (section 4.2) and our extended fast-mapping technique (section 3.3). Because the formulas are closed form, no sampling needs be done, which typically requires compromise between completeness and speed, especially when working in three dimensions [55].

All measurements and comparisons are done relative to the reference state, rather than the estimated vehicle state, thus references in the following text to "the vehicle" are more precisely interpreted as references to the current vehicle control reference state. This makes the algorithm deterministic given a set of goal inputs and obstacle locations, and is important to the performance of the algorithm because it decouples the trajectory generation from the control and estimation performance, which would otherwise cause undesirable feedback. One possible criticism of this technique is that the collision avoidance will not react to outside disturbances that drive the vehicle into hazards. The answer is that the controller is specifically designed to react to these disturbances and track the reference very robustly, so any additional coupling between the vehicle position and reference will cause unwanted control feedback, reducing the performance of the controller. This is the same methodology successfully employed by MIT in the DARPA Urban Challenge [38]

The first task in generating an avoidance trajectory is to determine how to handle the individual obstacles. As the algorithm loops through all obstacles in the map, the first check is whether the confidence in the obstacle estimate is above the defined threshold, otherwise that obstacle is ignored completely. If the obstacle is above the confidence threshold, the algorithm then checks if it is dangerously close to the

vehicle. Dangerously close is defined as the critical range, which was 68 cm + x_{stop} for our tests, where 68 cm is the 28 cm physical vehicle radius plus a 40 cm buffer to account for estimation and control errors, and x_{stop} is derived from Equation (4.11)

$$x_{stop} = \frac{1}{2}\frac{(\mathbf{v}_k \cdot \widehat{\mathbf{o}}_i)^2}{a_{\max}} \tag{4.15}$$

where $\widehat{\mathbf{o}}_i$ is the unit vector in the direction of the obstacle, the dot product with $\mathbf{v}_k$ computes the velocity in the direction of the obstacle. If the distance between the vehicle and the obstacle is less than the critical range, the trajectory generation is completely overridden with an acceleration command directly away from the obstacle of magnitude $a_{\max}$, referred to as an emergency reverse. Equipping this maneuver very nearly guarantees safety, but has poor performance, causing oscillations as the vehicle "bounces" off of obstacles. This emergency response is rarely used, however, because of the collision-corridor augmentation which acts as the primary collision avoidance protection.

The collision corridor is defined as the cylinder along the line from the vehicle to the goal with a radius equal to two times the critical range, and length from the vehicle location to the range of the goal, shown from a 2D plan view in Figure 4-7. The vector $\mathbf{d}_i$ is the shortest segment from obstacle i to the goal vector $\mathbf{g}$, which is computed using Equation (4.16)[1]. This vector serves a dual purpose, as its magnitude is the distance from the object to the center line of the corridor $\mathbf{g}$, and its direction is the ideal direction in which to accelerate in order to avoid the obstacle in three dimensions.

$$\mathbf{d}_i = (\widehat{\mathbf{g}} \cdot \mathbf{o}_i) \cdot \widehat{\mathbf{g}} - \mathbf{o}_i \tag{4.16}$$

$$\mathbf{o}_i = \mathbf{q}_i - \mathbf{x} \tag{4.17}$$

First, each obstacle is checked to see if it is in the collision corridor, which is true

[1]A diagram of this geometric construction is presented in Figure 4-3.

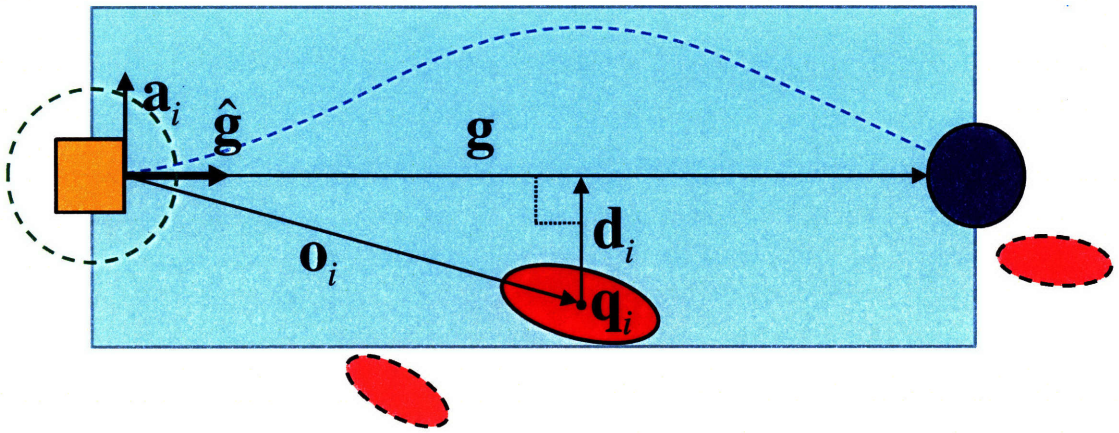


Figure 4-7: Collision corridor with two obstacles outside the corridor ignored (dashed ellipses). The green dashed circle around the orange vehicle square is the critical range used to initiate emergency reverse maneuvers. The corridor is a three-dimensional cylinder, flattened in these diagrams to facilitate discussion.

for obstacle i when all of the following are true

$$\begin{array}{rcll} \|\mathbf{d}_i\| & < & 2\,(68\text{ cm} + x_{stop}) & \text{(within the corridor radius)} \\ \|\mathbf{o}_i\| & < & \|\mathbf{g}\| & \text{(closer than the goal)} \\ \mathbf{o}_i \cdot \mathbf{g} & > & 0 & \text{(in front of the vehicle)} \end{array} \tag{4.18}$$

For each obstacle in the corridor, a cross-goal reaction acceleration direction $\mathbf{a}_i = \widehat{\mathbf{d}}_i$ is computed according to Equation (4.16), which are summed together, each weighted by the distance to the goal vector $\|\mathbf{d}_i\|$ and the inverse of its time to impact t_i. This weighting prioritizes features that are in more immediate danger of collision (based on t_i), and have a more clear optimal reaction direction (the nearer an obstacles is to the goal vector, the more directly in front of the vehicle it is and the less important which direction you choose to go). This sum is normalized to determine the aggregate reaction acceleration direction $\widehat{\mathbf{a}}_a$. Because all of these components are perpendicular to $\mathbf{g}$, the aggregate $\mathbf{a}_a$ will be as well. This perpendicular acceleration causes the vehicle to move sideways, around obstacles, while maintaining its momentum toward the goal, resulting in smooth and efficient trajectories.

The magnitude of this acceleration component is defined to be 1.5 times the distance to the obstacle with the minimal time to impact, denoted by subscript min, divided by $t_{min}\tau_r$, with $\tau_r = 1$ s in our tests (which empirically shows to generate good trajectories), thus the resulting reaction acceleration component $\mathbf{a}_r$ is

$$\mathbf{a}_r = 1.5\frac{\widehat{\mathbf{o}}_{\min}}{t_{\min}\tau_r}\widehat{\mathbf{a}}_a \tag{4.19}$$

$$\mathbf{a}_a = \left[\sum_{i=1}^{n}\frac{\|\mathbf{d}_i\|\mathbf{a}_i}{t_i}\right] = \left[\sum_{i=1}^{n}\frac{\|\mathbf{d}_i\|\widehat{\mathbf{d}}_i}{t_i}\right] = \left[\sum_{i=1}^{n}\frac{\mathbf{d}_i}{t_i}\right] \tag{4.20}$$

$$t_i = \frac{\|\mathbf{o}_i\|}{\widehat{\mathbf{o}}_i \cdot \mathbf{v}} \tag{4.21}$$

This reaction acceleration component is added to the goal attraction and orbital cancellation components to generate the final acceleration vector $\mathbf{a}_k$, as defined in Equation (4.1).

4.3.1 Collision Avoidance Observations

This collision avoidance scheme has proven to work well in general, but unusual environments can cause odd behavior. The most challenging situations for a standard potential function based reactive obstacle avoidance algorithm are local minima [5], formed by obstacle "nets." Imagine a group of four symmetrical obstacles arranged in a square located in a vertical plane directly between the vehicle and the goal, shown from a 2D plan view in Figure 4-8. Obstacles 1 and 2 are lined up on top of each other (denoted $\mathbf{q}_{1/2}$), as are 3 and 4 ($\mathbf{q}_{3/4}$), such that any reactions out of the plane cancel out from symmetry.

In a typical potential field solution, the four potential functions will form a minimum between each other that will trap the vehicle, unable to plan its way out. Using the new corridor method, the reaction vectors will approximately cancel out in Equation (4.20), but any slight asymmetry (which is guaranteed in a real system) is re-scaled by Equation (4.19) to cause a full reaction favoring initial motion (we'll assume to the left). As the vehicle accelerates toward obstacles three and four, they will get closer to the goal vector, decreasing $\|\mathbf{d}_{3/4}\|$, and will thus be weighted less

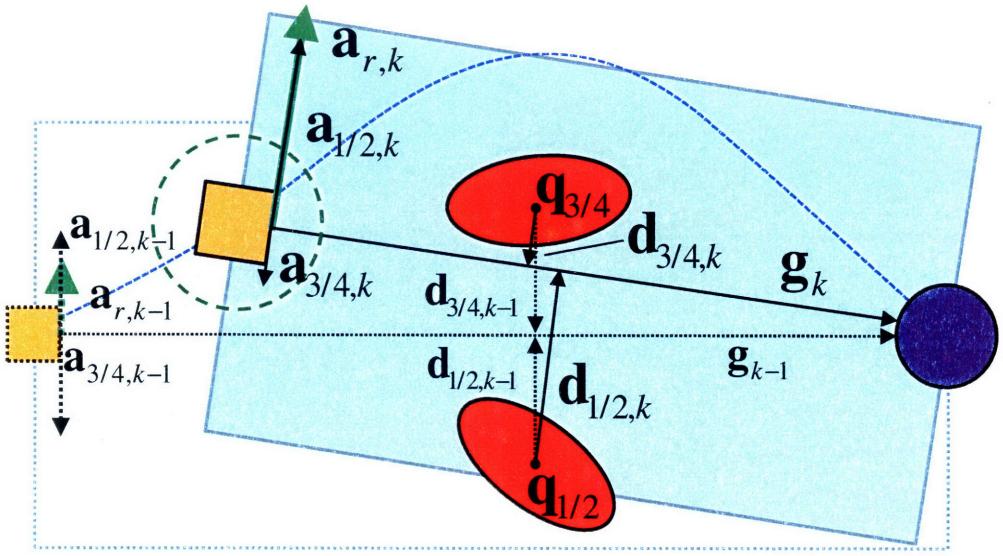


Figure 4-8: Top view of potential obstacle "net" situation. At time $k-1$ the effects of the obstacles almost perfectly cancel, but the vehicle still reacts intelligently.

heavily. This compounds the original imbalance, pushing the vehicle more in the same direction, out and around the "net" that might have otherwise trapped it. If the obstacles are spaced just wide enough that one of them goes out of the corridor before the vehicle makes it all the way around the other (as shown in the figure), the vehicle will have a tendency to "pop" back into the center, oscillating back and forth as it moves forward, seemingly trapped. This is an unusual and potentially catastrophic case, but because the corridor is two times the radius of the critical range, the vehicle will actually center itself between the obstacles and just barely fit through them. The second level heuristic protects the vehicle from collisions as it squeezes through. The ultimate result of this combined repulsive/corridor cross-goal acceleration is a closed-form goal pursuit and obstacle avoidance heuristic which runs in linear time with the number of features in the environment.

Figures 4-9, 4-10 and 4-11 show simulated 2D flights around around different arrangements of obstacles. The green dashed circles are buffer zones used for emergency maneuvers, and the red dashed circles are collision zones defined by the physical size of the vehicle. The purple orb is the goal, and the blue trace is the simulated reference

trajectory.

Compared with typical reactive planners, the corridor augmentation we have presented has several advantages. The first is that it generates smoother and more natural trajectories, tending to move the vehicle directly toward the outside edges of in the way obstacles, just as a person would. The second is the reduced risk of getting trapped in a minimum. Figure 4-12 shows an example comparing the results of using the emergency reverse only, which is similar to a closed-form artificial potential function repulser solution, to the same emergency reverse algorithm augmented with the corridor system. 20 features placed in a plane are used for visualization purposes as it is hard to judge performance of 3D flights in 2D plots. 20 features in the flight plane is roughly the same density as 100 features in the flight volume.

Computing the effect of the corridor increases the solve time to about 1.5-2 times the potential only solutions, which is considered a worthwhile penalty for the gain in the performance, especially since the repulsion solution is incredibly simple and fast to begin with, but consequently gets stuck before reaching the goal. The corridor algorithm operating on 100 features in three-dimensions runs unoptimized at about 800 Hz, which is on a fast desktop, but in a slow Matlab environment. This is nearly two orders of magnitude faster than the 15 Hz at which the vision is updated, and is primarily a result of the simple closed-form equations. In the C++ environment, the time to compute the obstacle avoidance is almost immeasurable, clocking at an average 1 μs with a typical 50 features of high enough confidence to be used in the avoidance calculations.

4.4 Closed-Loop Vision Flight

The primary challenge presented by closed-loop vision flight lies in the interaction between the flight performance and the state estimate, addressed previously in section 3.2. Without vision feedback, the vehicle is capable of maintaining a quasi-stable hover by damping rotations with gyro feedback and fusing the gyros and accelerometer to track the gravity vector. Unfortunately this flight mode is not sufficient for

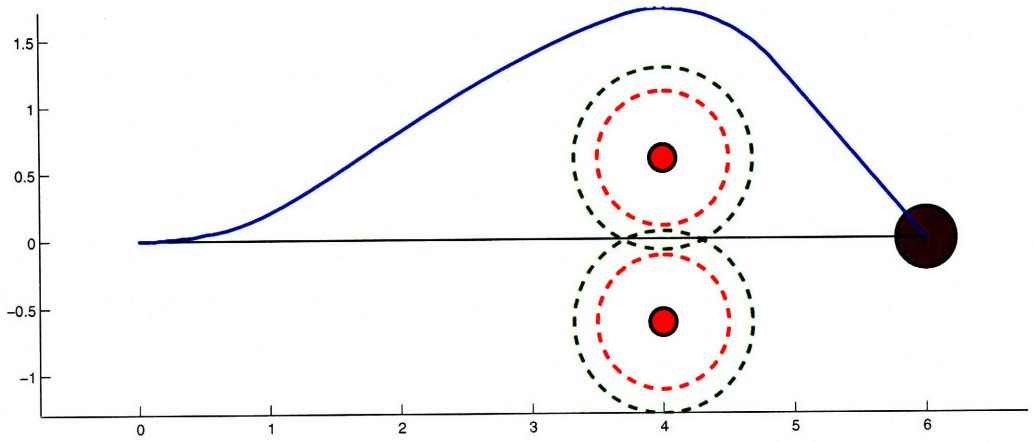


Figure 4-9: A typical scenario in which a clump of obstacles are detected and smoothly avoided. A typical potential function solution would get stuck in the notch between the obstacles.

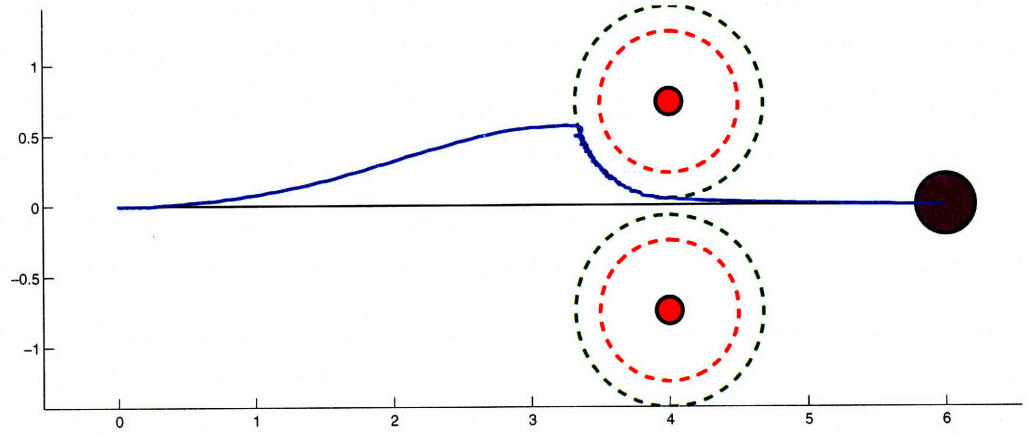


Figure 4-10: A very unusual scenario with a symmetric obstacle "net". The vehicle gets a little confused, but ultimately finds the gap between the obstacles using the emergency avoidance routine.

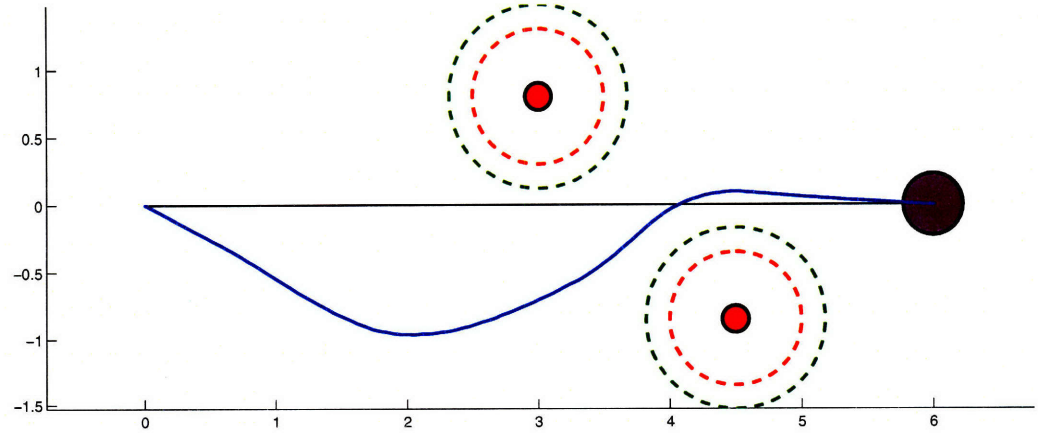


Figure 4-11: A little more unusual arrangement with staggered obstacles. Because the vehicle places higher weight on obstacles with sooner time to impact, it avoids the nearer obstacle first, slaloming between them.

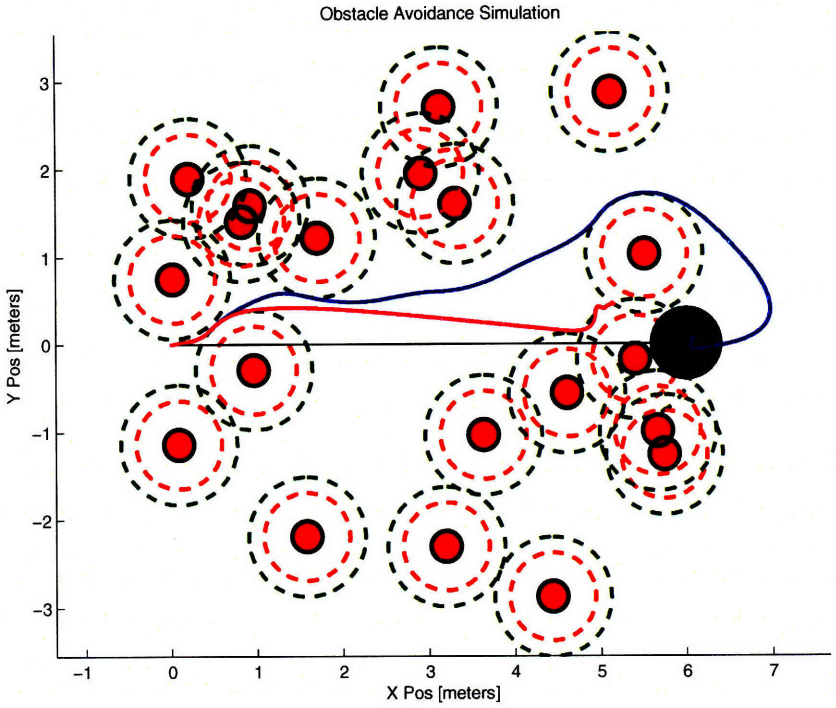


Figure 4-12: A comparison of repulsion only trajectory in magenta vs. the corridor augmented technique in blue with 20 obstacles in the plane $z = 0$.

our goals because disturbances in the environment will eventually cause the vehicle to drift off into an obstacle and crash. We also do not want to simply hover, but rather explore the environment.

Adding vision enables the vehicle to estimate the absolute position, which then can be used to close the control loops. From a single point of view, the vision features stay approximately unchanged and the estimate is stable. However, as we add accelerations to move around the environment, our point of view changes; features move and are tracked, but also go out of view and must be replaced, and occlusion and reflective surfaces cause complex vision scenarios with false corners that do not correspond to physical points in 3D space, making the feature tracking and estimation significantly more challenging.

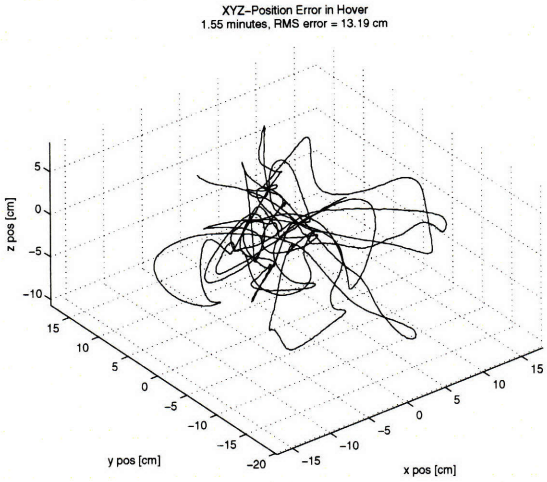
With persistent features it is possible to generate a drift-free estimate, but as features go out of view, they must be replaced and estimation errors will accumulate. These inaccuracies can cause the vehicle to act aggressively, as if step functions had been fed into the controller. Aggressive motions cause the images from the onboard

camera to blur and go out of the camera field of view more frequently, compounding the rate of estimate error propagation in a positive feedback loop. Using a low gain on position error solves this problem by having the vehicle react smoothly to step changes in the position estimate, and a large integrator component keeps the tracking performance high, but with a slower time constant. High integrator gains typically cause problems with overshoot, but, because of the error in our estimate, overly-precise tracking is not important. It is also interesting to note that the accuracy of mapping estimates increases with larger visual parallax, so slow oscillations driven by a large integrator have a tendency to improve the vehicle's ability to resolve feature locations.

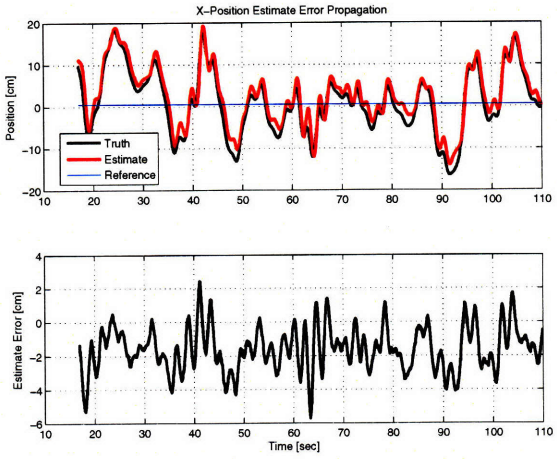
4.4.1 Drift-Free Hover Results

When hovering, the vehicle maintains an almost fixed point of view, maximizing feature persistence and minimizing drift. Figure 4-13 shows a typical hover using the vision-based state estimate for control feedback. Because the vehicle remains in the same approximate location, no features are lost and thus the average drift for the 96 seconds the vehicle is commanded to hover is negligible.

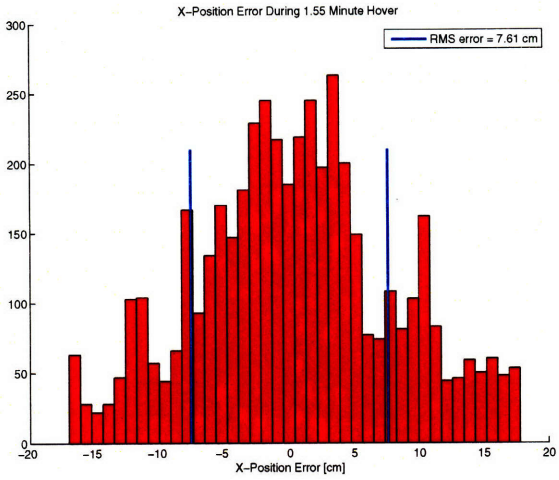
Figure 4-14 shows the result of the obstacle avoidance flying the physical vehicle around simulated obstacles. The vehicle is simply commanded to go the opposite side, then return, and avoids the virtual hazards autonomously.



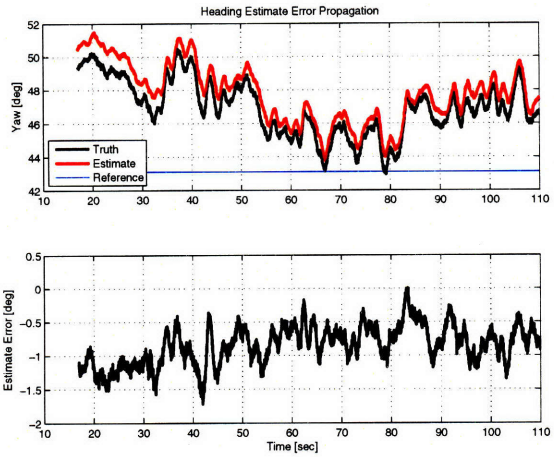
(a) 3D plot of flight.



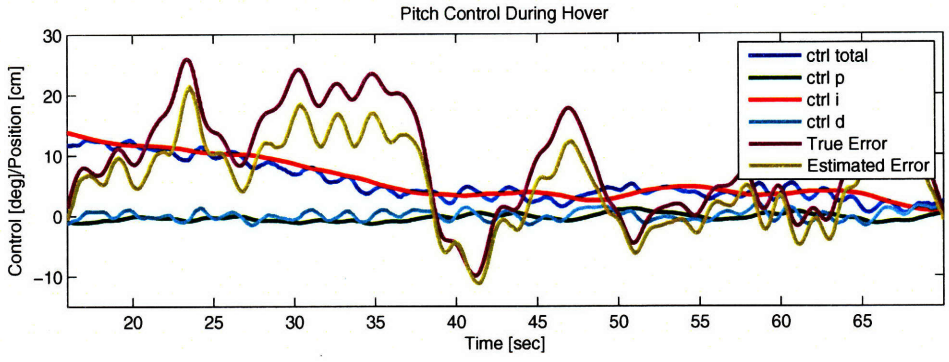
(b) Propagation of X position estimate error during hover.



(c) Distribution and RMS error in X truth position deviation from reference.



(d) Yaw estimate error propagation during hover.



(e) Control input contributions.

Figure 4-13: Simple hover with flight control loop closed using vision-based state estimate for full-state feedback.

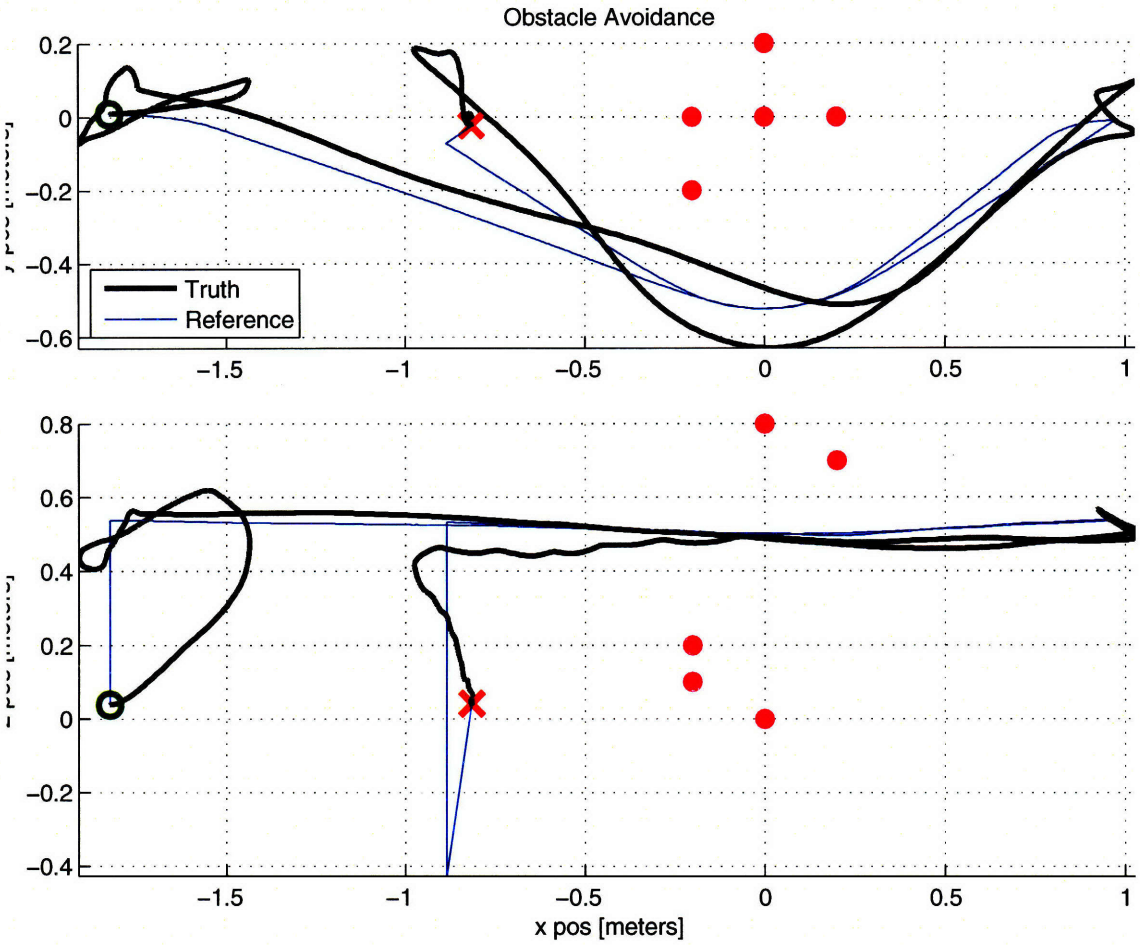


Figure 4-14: Real flight around simulated obstacles.

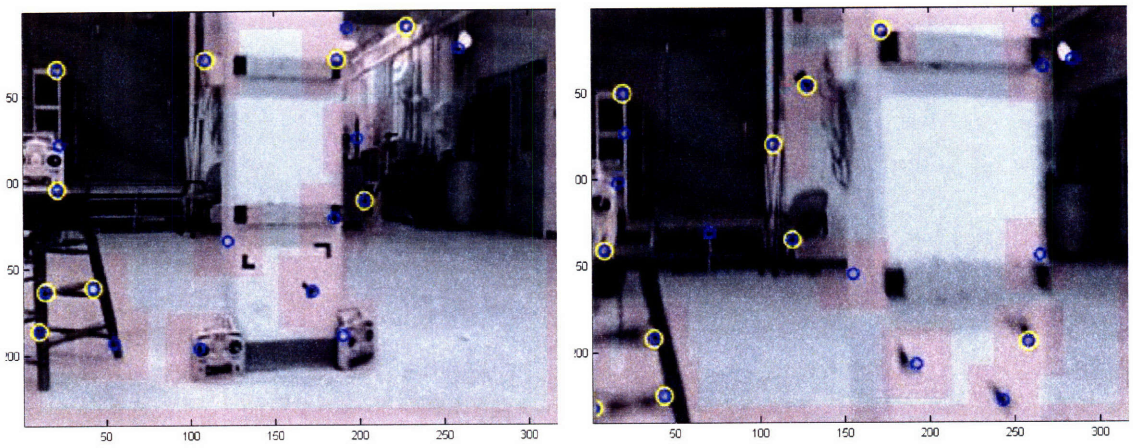
Chapter 5

Results

The section includes the results of the combined algorithms discussed in the central chapters, which include sub-results of their own. Computational requirement results are also presented in order to build the case that these algorithms could likely run on an embedded computer.

5.1 Vision-Based Control and Obstacle Avoidance

These tests are the culmination of all the developments of this thesis. Vision-based state estimation with emulated inertial data from the Vicon camera system is used to generate the state estimate, which is used for full-state control feedback to stabilize the vehicle, track the reference trajectory, and map the environment. The mapped features are processed with the reactive obstacle avoidance algorithm, and a single waypoint entered on the opposite side of the room is loaded into the system manually by the operator. The finite-acceleration trajectory generator integrates a smooth trajectory toward the waypoint, taking into account the orbital cancelation and obstacle avoidance reactions, safely guiding the vehicle around the clutter directly blocking its path to the goal. No information is known about the pole, stool, and other clutter – it is perceived entirely with the single camera onboard the vehicle. The position, velocity, and attitude data from the Vicon tracking system are only used for visualization and post-processing all the way from take off to landing.



(a) The environment

(b) Going over the stool.

Figure 5-1: Onboard camera view of the final environment setup for autonomous obstacle avoidance.

Figure 5-1 gives a sense of what the environment looks like to the vehicle. It is low resolution, gray, and blurry, just as an onboard micro-camera is expected to be when pushing the limits of small-scale flight. The feature tracking is sufficiently robust to help the vehicle do what it needs to do in order to traverse the environment autonomously. The environment was set up with the pole and additional clutter in the center of the room in order to give the vehicle some room to maneuver, but with a clear and complex obstacle set blocking the direct path to the goal.

Figure 5-2 shows two Matlab plots from the top (x, y) and side (z, y) of a successful obstacle avoidance flight. These plots serve as a tool for quantitative performance analysis with physical units. Over the course of the flight, the vehicle travels 5 meters to the opposite side of the room with an ultimate error of about 1 meter, most of which was accumulated near landing because of the sparseness of the wall behind the pole, making for difficult feature tracking. The green circle is the take-off point, the red line is the estimated trajectory, and the purple spots are 3D points the vehicle has mapped in the world based on its vision and ego-motion estimates. The blue block and green cylinder are hard-coded graphics of the physical pole and stool, and the black line is the truth trajectory from the Vicon external camera system, which are collectively intended for judging the ego-motion estimate and mapping accuracy.

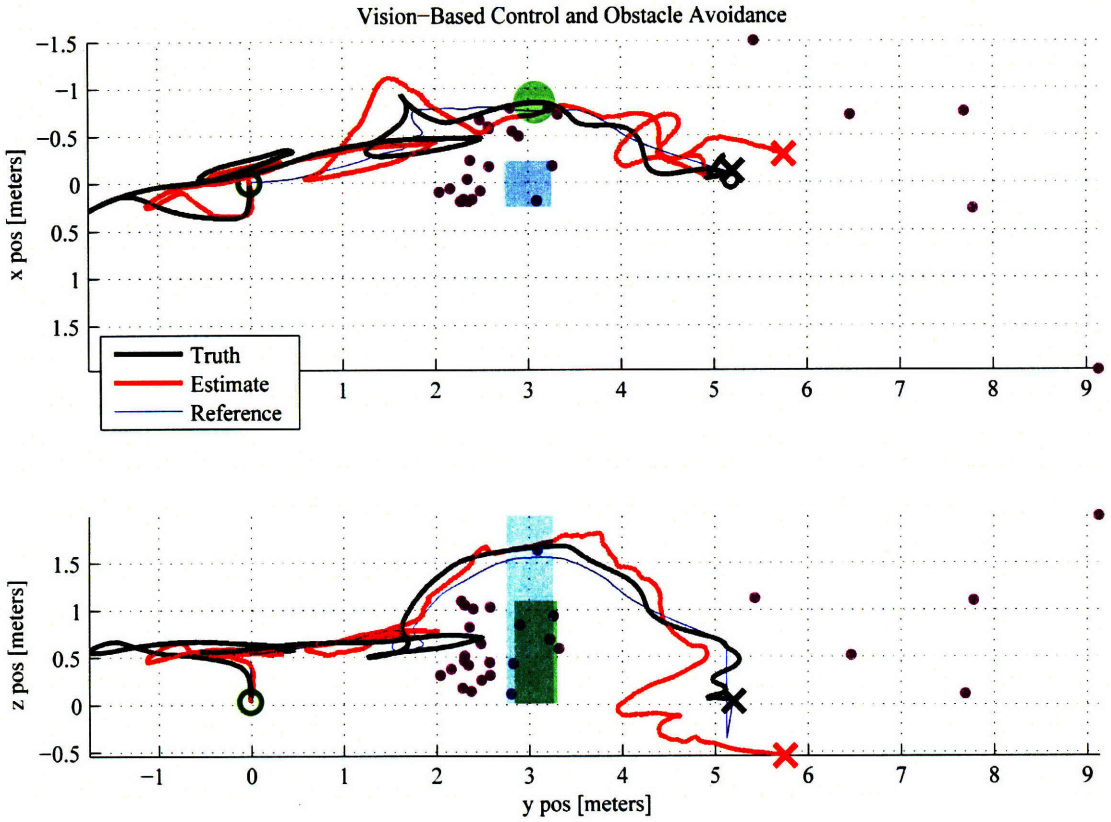


Figure 5-2: Orthographic views of flight over stool.

The purple spots represent a snapshot in time of the points used to compute the three-dimensional reactive obstacle avoidance acceleration components, which were integrated over the course of the flight to create the blue reference trajectory the controller was tasked with tracking. As the vehicle attempted to move toward the goal, the obstacle avoidance decided to fly the vehicle up, around the pole to the left, over the stool, and down to a smooth touchdown near the goal. Presumably the odd path was chosen because of the non-deterministic nature of the map formation, because the obstacle position estimates shift with variations in vision measurements and state estimation.

Figures 5-3 and 5-4 are more visually intuitive 3D perspective screenshots taken of the real-time operator visualization and input application from two different view angles and points in time during an offline playback of the flight data from Figure 5-2. The color coding is similar to Figures 5-2 and 5-6, with the red trace being the

Vision/INS position estimate, the red ellipsoids representing the mapped obstacles, and the black trace indicating the time history of the true vehicle position from Vicon. Again, the yellow block and brown cylinder were added manually in order to give a baseline for judging accuracy of the mapping and proximity of the vehicle to physical obstacles.

Figure 5-5 is a linked video of the flight, composited from several replays of the flight from different perspectives, including a video camera that captured the physical flight, and the processed onboard video showing feature tracking (small blue and yellow circles) and feature culling (large blue, green, and red flashes). Note how difficult it is for the system to track the image features when the vehicle lands because of the aggressive motion that blurs the image. These aggressive motions are unavoidable when impacting the ground, but have were essential to avoid in the control design because they would decrease the estimate performance. The result is that the flight is very smooth because of the decreased proportional gain, and tracking performance regained with a high integral component, which also causes the low-frequency oscillations that help the vehicle build parallax on the observed obstacles.

Several tests were done in this particular setup as the various parameters in the estimation filter and obstacle avoidance were tweaked to get good performance. In general we found that the system was quite robust and capable of maintaining stable flight with safe obstacle avoidance behavior with a fairly wide range of parameters. This is because the general formulation of the obstacle avoidance and estimation filter are designed to be robust to complications inherant in our flight scenarios. The primary advantage is use of relative obstacle avoidance which allowed the vehicle to avoid obstacles effectively after any amount of accumulated drift. Figure 5-6 shows a plot of a representative flight with the same starting condition and goal state as the last flight, presented in Figure 5-2. This time the vehicle chooses a much more sensible path around the pole to the right, avoiding the stool entirely. The data visible in the figure is a clear example of the complimentary offset errors that develop in both the feature estimates and the estimated vehicle position, as discussed in section 3.3.3. Looking at the upper (x,y) plot of Figure 5-6, notice how the estimated

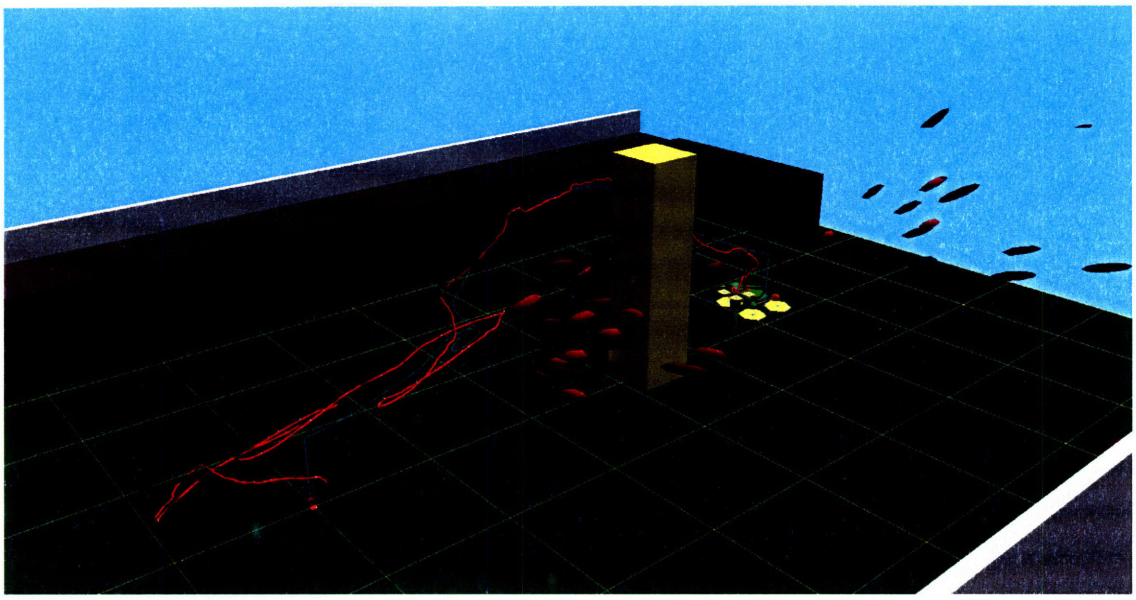


Figure 5-3: 3D visualization of flight over stool from back of room.

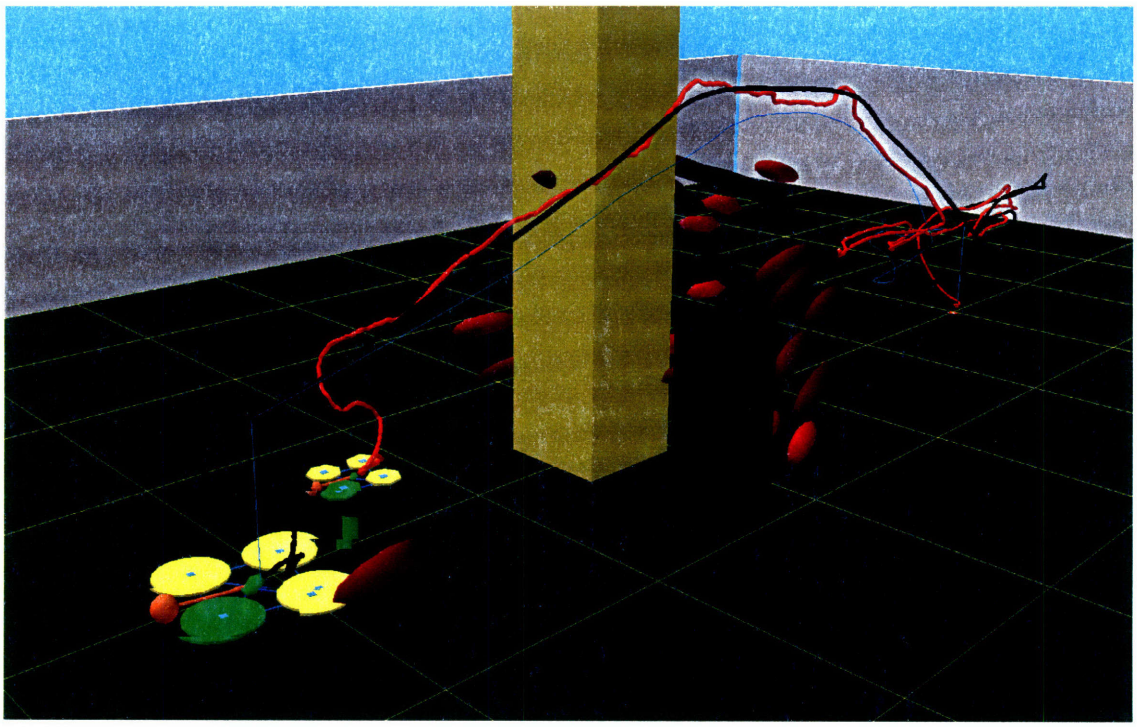


Figure 5-4: 3D visualization of flight over stool from operator perspective.

vehicle path is offset about 0.2 meters to the left of the truth trajectory as the vehicle approaches the the point (2,0) near the pole. It is also clear in the figure that the feature estimates are approximately 0.2 meters to the left of where the pole actually was (represented by the light blue rectangle) as well. Even though the true position of the vehicle was much closer to the axis (3,0,z) where the pole physically exists than the collision avoidance thinks it was, it does not matter because the avoidance system is generating the trajectory based on its own estimate of the pole, as described in section 4.3. Thus the resulting trajectory, minus the error in the vehicle estimate (and thus tracking) causes the vehicle to fly around the pole with a real-world buffer range close to the 68 cm designed in the original obstacle avoidance. In fact, were the location of the pole hard-coded into the obstacle avoidance algorithm, the vehicle would probably crash because the error in the estimate would not be countered by the complimentary error in the obstacle estimates. Figures 5-7 and 5-8 are more visually intuitive screenshots of this flight.

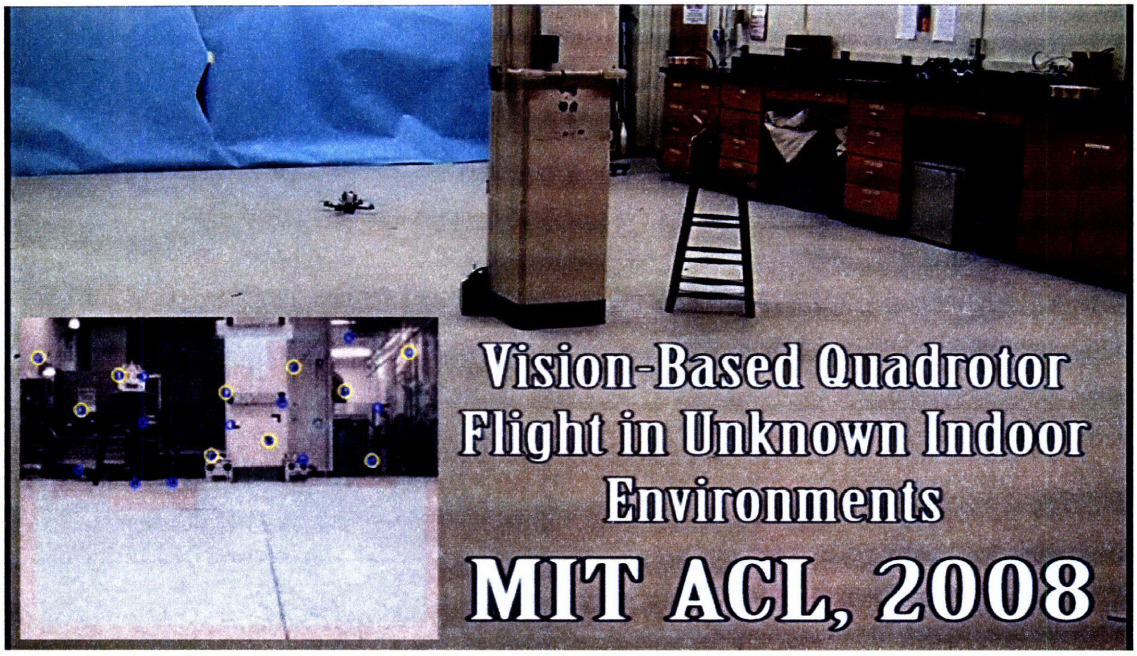


Figure 5-5: Autonomous vision-based control and collision avoidance flight video, available at acl.mit.edu.

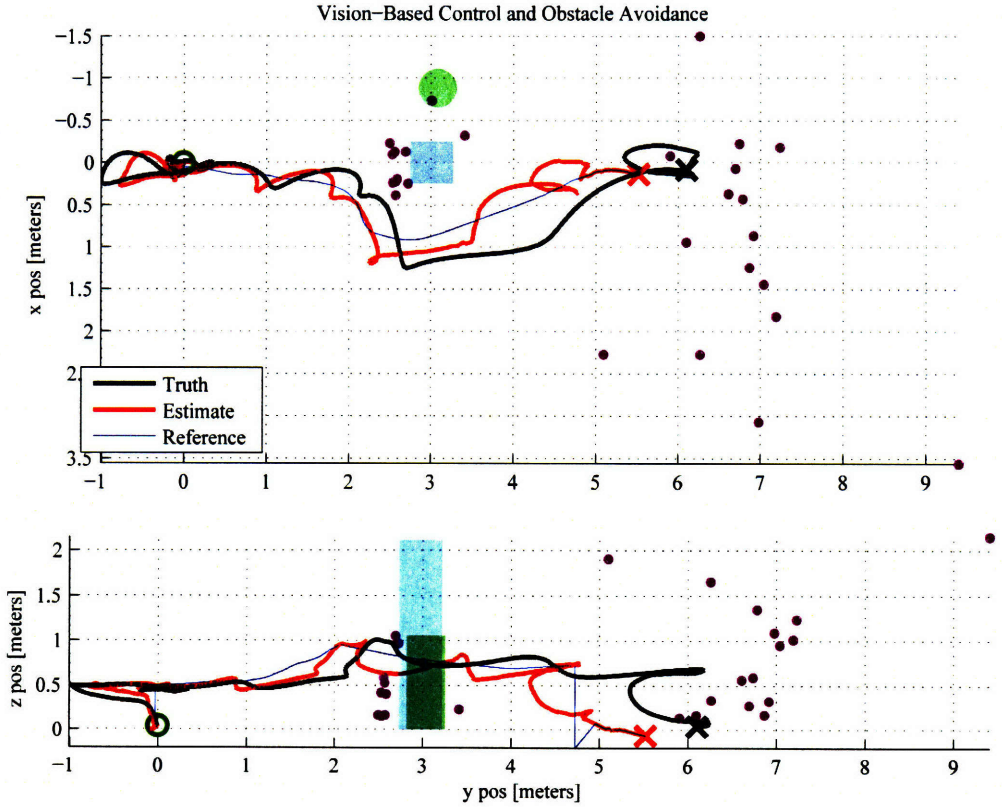


Figure 5-6: Orthographic views of flight around the stool and pole.

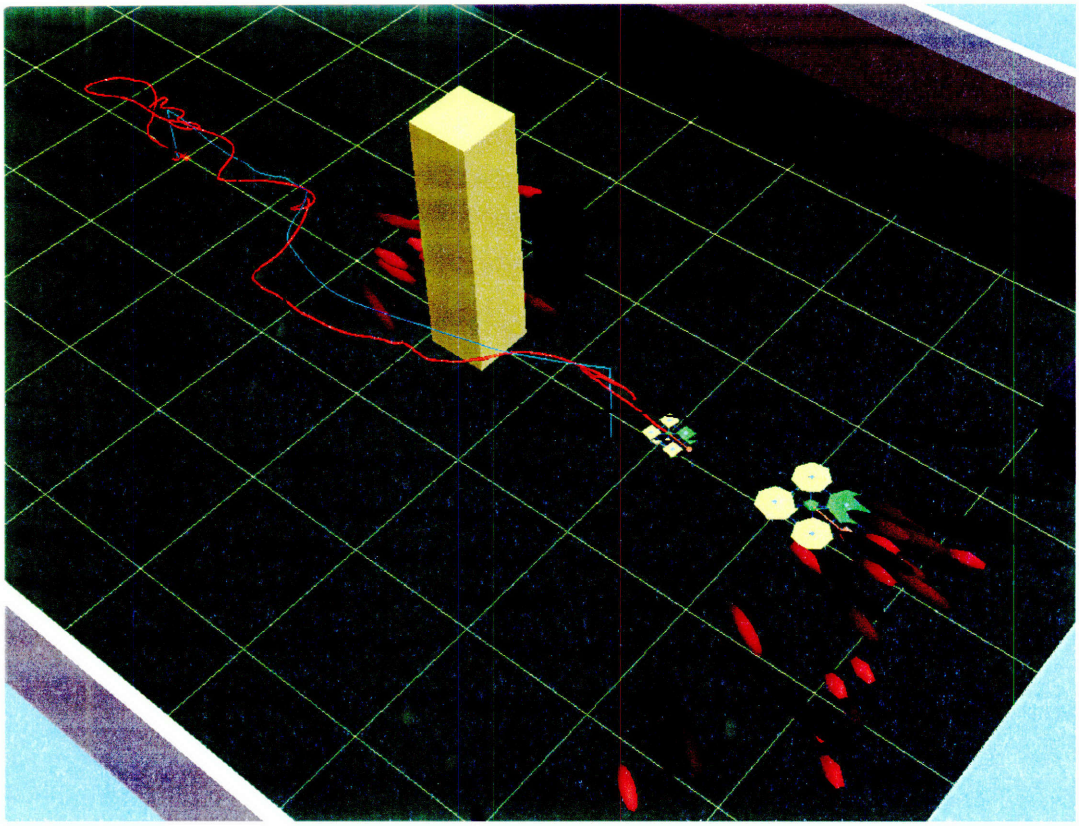


Figure 5-7: 3D visualization of flight over around pole and stool from high in front.

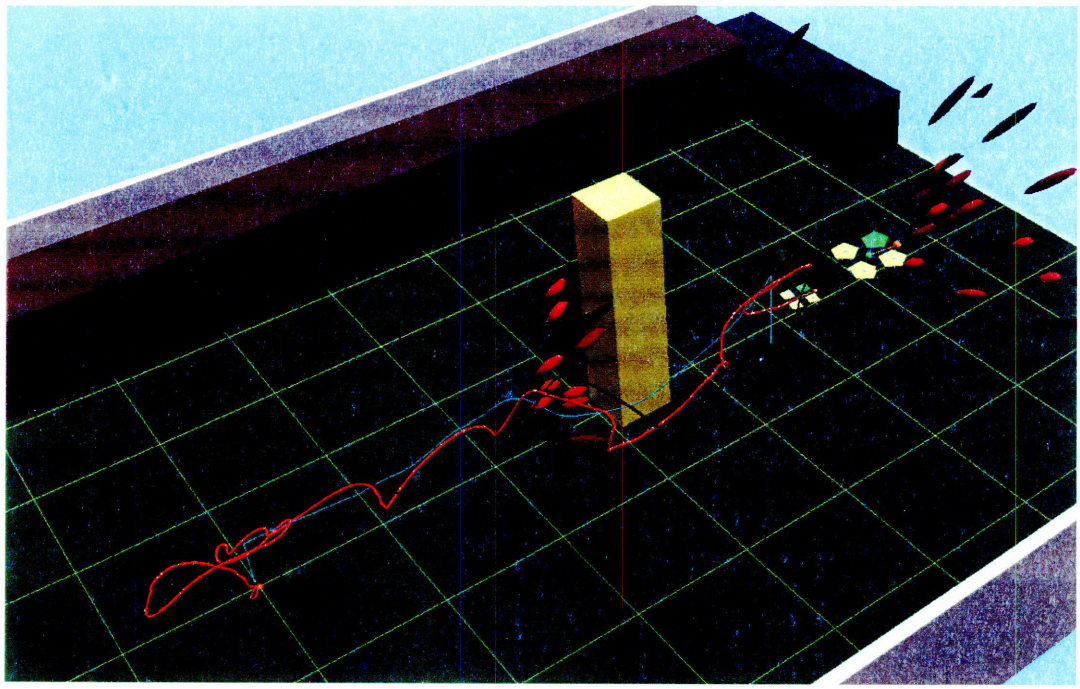


Figure 5-8: 3D visualization of flight over around pole and stool from high and behind.

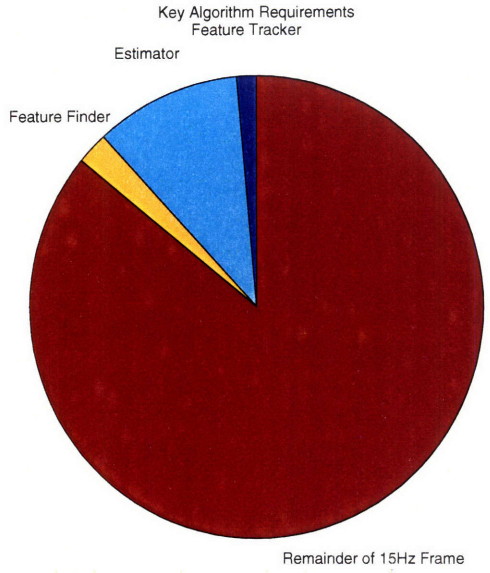
5.2 Computational Demand Analysis

Timing is done relative to the fixed time allotted per step based on the 50 and 15 Hz inertial and vision cycles (20 ms and 67 ms time periods), respectively. In order to analyze the computational requirements of the algorithms, the offline data replay system was employed such that the subroutines could be run in series in a single thread to maximize timing precision. The processor used was a Core2 Quad Q6700 at 2.66 GHz with 4 GB of RAM, for the reasons described in section 2.2. The proportionate and absolute time spent on the various tasks are listed in Table 5.1 and depicted in Figure 5-9.

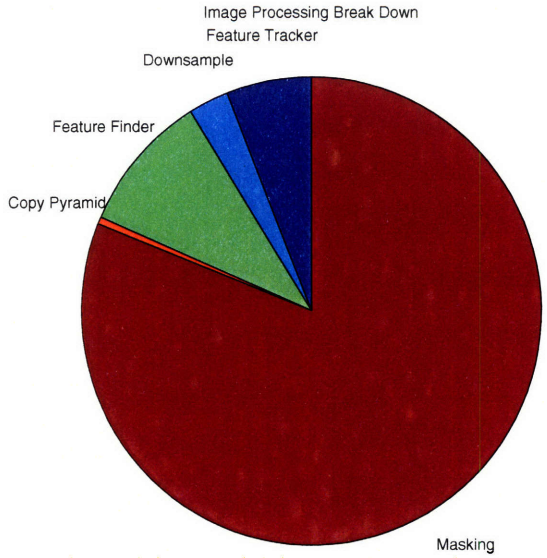
The total average computational demand of one second of operation (not including masking time, presuming that it can be replaced with a more optimal solution) with vision at 15 Hz and inertial data at 50 Hz is 141 ms, or 14.1%. The Q6700 processor is measured to execute 49,161 Million Instructions Per Second (MIPS) [65], but because this would be using all four cores, we divide by 4 to get the power available to our single-threaded playback system—12,290 MIPS. 14.1% of this is 1,732 MIPS, which is what we are using for our core algorithms. This implies that the system could run on a significantly slower embedded microcontroller, such as the new Intel Atom processor

Table 5.1: Breakdown of computational demands for key elements of the onboard flight system.

Process	Average Time Needed	Range
Vision	67 ms	
Masking	12.7 ms (19.1%)	[12.2, 18.6] ms
Feature Finder	1.51 ms (2.27%)	[0, 11.5] ms
Feature Tracker	0.93 ms (1.4%)	[0.75, 2.10] ms
Vision Update	6.80 ms (10.2%)	[4.31, 65.2] ms
Mapping	4.2 μs (-)	[3, 5] μs
Avoidance Update	0.55 μs (-)	[0, 1] μs
Inertial	20 ms	
Time Update	37.9 μs (0.19%)	[36, 60] μs
Onboard Guidance	0.28 μs (-)	[0, 1] μs
Control	1.9 μs (-)	[1, 6] μs



(a) Primary system algorithms compared to time available with 15Hz video.



(b) Timing breakdown of image processing algorithm. Image masking clearly needs to be replaced with a more efficient option.

Figure 5-9: Breakdown of time spent performing various tasks. The primarily inertial functions were insignificant($\sim$ 40 μs), and the key vision processing algorithms used only about 14.1% of the 67 ms available per video frame.

for mobile devices [29], or the ARM Cortex-A8 which has a claimed execution rate of 2000 MIPS at 1 GHz with the floating point NEON media coprocessor, while consuming less than 300 mW of power [44].

Note that the maximum 65.2 ms on the vision update is very near the 67 ms per frame, thus would exceed the time available if running on an embedded system. This peak is from repeatedly initializing the covariance of newly seen features as the vehicle lands abruptly and the image is shaken beyond usability, notable at the end of the video in Figure 5-6 and the spikes at 38 seconds of Figure 5-11. The covariance initialization is also in the process of further optimization by Draper Labs to reduce the associated time. Occasionally taking longer than the allotted time is ok—as long as the average is less, time can be borrowed from adjacent time steps without much consequence, shown by the fact the system can operate with variable and inconsistent camera rates due to transmission delays in the wireless system, and changing light levels, ranging from 12-30 Hz [24].

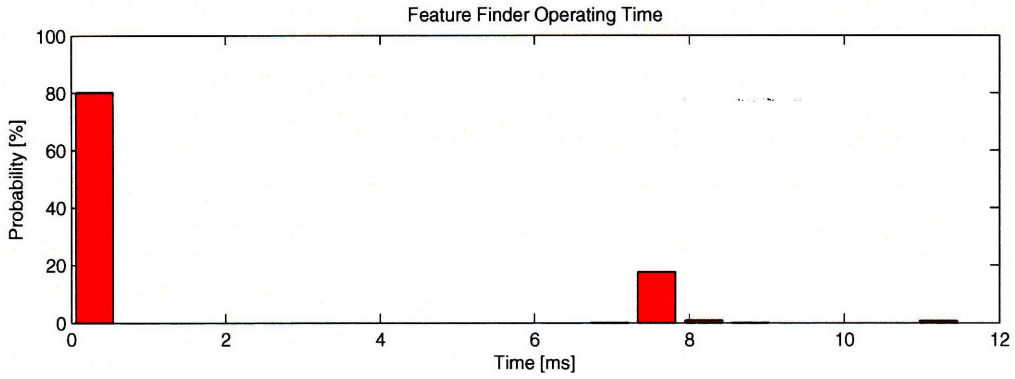


Figure 5-10: Probability histogram of how long the the feature finder will likely take.

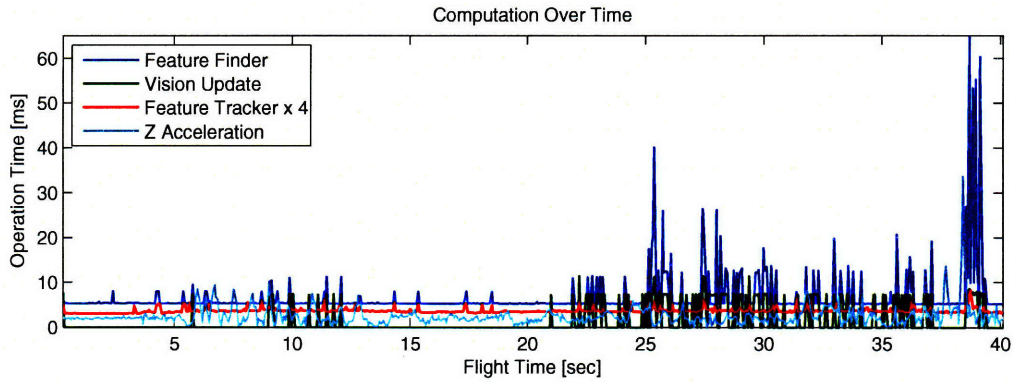


Figure 5-11: Processing time of finding and tracking features, conducting vision updates, and current Z acceleration as the flight progressed.

Also note the high max on feature tracking relative to its average—this is because the feature tracking is only called when features are lost, and thus is not called at all for 80% of vision updates in most flight tests (almost never for hover tests). Of the instances the system does need to find new features, the `goodFeaturesToTrack` algorithm is very consistent, and takes between 7.0 and 8.2 ms 95% of the time, as shown in Figure 5-10. This is because the algorithm methodically scans every pixel of the image, except those masked out, and thus takes a very consistent amount of time. There are some unexplained outliers near 11 ms that are infrequent enough to disregard, thus the feature finding step can be relied upon to almost always operate in under 8 ms on this hardware platform, independent of the imagery.

Figure 5-11 shows the correlation between finding features and the time needed to process vision updates. At the end of the flight, the vehicle has a bit of a rough landing, indicated by the high Z acceleration. This causes the imagery to blur, with

noticeable strain on the feature tracker, and tremendous strain on the Vision filter, which spikes all the way up to 60 ms from its 6.8 ms average. During experiments, it is possible for the filter to get into an slow state like this while still flying, which can slow down the entire system to below the operable limits, causing the vehicle to crash. This is another positive feedback loop as described before in section 4.4, as the slow updates will decrease the control performance, which causes the vehicle to oscillate, blurring the imagery further, compounding the problem. The long solve time is presumed due to poor filter convergence in the face of bad measurements. For a robust solution, a fix would have to be put in place to limit the amount of time the filter can spend on one update.

The measurements of onboard guidance and control were insignificant compared to the primary computation required by the image processing and state estimation. State estimate time updates based solely on inertial data are also very fast, clocking in at about 36 μs (or about 0.2%) of the 20 ms update period available), and thus the vehicle had no trouble at all keeping up with the IMU. The image processing was a measurable portion of the 67 ms vision cycle period, with the `goodFeaturesToTrack` feature finding taking an average of 1.5 ms per frame (2.2%) and the Lucas-Kanade feature tracking demanding about 0.93 ms per frame (1.4%) on average.

The vision based state update was slightly more of a burden, taking an average of 6.8 ms (10.2%). The masking algorithm is a clear sign of weakness and could easily be replaced with a Rectangle Of Interest (ROI) method, greatly increasing performance. An ROI could also improve the feature spread by selectively search for features in less well covered areas of the image. The reason the masking algorithm is so slow is because of the OpenCV implementation of matrices which use a clunky memory interface to access individual pixels, and a very slow erode function which masks out the pixels surrounding already masked out pixels, slowly growing the masked areas to the desired size. Although these particular functions could be improved, it would make more sense to entirely switch to an ROI method of spreading features.

Chapter 6

Conclusion

Arbitrary 3D navigation in complicated environments is a challenging task with many valuable rewards. The general problem has been approached from many different directions with much success, and this field of research is growing exponentially in academia, private, and government sectors. There is significant cross-over of knowledge and technology between different robot paradigms, from wearable computers to submarines to micro-helicopters.

We have presented several pieces of a proposed solution, and demonstrated physical flight results supporting their validity. The ego-motion and environmental estimation based on monocular vision data and inertial measurements proves to be very promising in mostly static environments with high degrees of clutter. The trajectory generation and associated control techniques have demonstrated ability to avoid obstacles in unknown environments autonomously with minimal computational effort. The concept of hybrid control is very promising, using a combination of autonomously generated, high-level waypoints, interpolated with the high speed onboard trajectory planner and reactive obstacle avoidance, making the vehicle capable of executing advanced autonomous missions while remaining safe in the event of communication loss (or operator neglect). Although the complete benefits of this ability have not been demonstrated extensively, the very sparse input sets given in the ultimate obstacle avoidance tests demonstrates the ability of the onboard planning system to get from point A to point B effectively all on its own, even when the direct path is far from

clear.

The resources of the RAVEN rapid-prototyping testbed have also been demonstrated as an incredibly useful tool for development of sensing-based algorithms. The high-fidelity state information provides excellent performance feedback, as well as a safety net for protecting the vehicle while developing new and unstable algorithms in hardware with minimal risk of vehicle damage.

Finally, the road has been set for true embedded implementation of the onboard algorithms. The onboard system has been shown to be very computationally efficient, even in the very open environment of hovering flight in three dimensional spaces with upwards of 100 mapped obstacles. Many of the hardware challenges will actually diminish with the move to embedded processing, specifically because of the current performance losses due to wireless transmission of visual data. Embedded cameras can provide raw imagery without the compression artifacts and delays, transmission irregularities, and other hindrances we have had to endure in our testing. The move to true embedded sensors will also potentially benefit the algorithm performance as there will be reduced delay and higher fidelity with a similar noise level. Ultimately, a fully autonomous, self-contained HMAV vehicle will almost certainly be developed in the near future with capabilities exceeding those we have demonstrated here.

6.1 Future Improvements and Extensions

Unfortunately, vision-based sensing suffers heavily when confronted with amorphous objects, highly specular surface, moving hazards, and visually plain un-textured environments. Binocular vision is a logical next step, and can also be beneficial to the remote operator, providing stereo vision for more immersive real-time environment feedback. Stereo vision solves the problem of moving and deforming obstacles as it can estimate range in a single time step, but stereo vision traditionally has issues with additional computation, double the camera hardware and related power and weight, synchronization, and calibration issues. A potential solution to this problem would be to explore optical lens separation techniques in order to split a single camera im-

age into two different points of view. This would guarantee image synchronization and identical lens distortion properties, solving many of these challenges, but stereo vision still suffers the challenges attributed to vision in general, such as specularity and visually plain environments.

Adding LIDAR capabilities to a vision equipped system can provide a similar capability as stereo vision in estimating range and thus full position of obstacles in one time step, giving the vehicle the ability to avoid moving hazards, but again adds significant weight, size, and power penalties. However, as it is miniaturized, LIDAR could prove an even more valuable augmentation than stereo vision as it can potentially operate in a wider range of environments, robust to changing ambient light levels and environments with low or zero texture [32].

Flying strictly indoors is also a large limitation of our system that can be easily extended with existing technology. The emergence of micro-GPS systems and high performance 3D compasses can give HMAVs very high capabilities in outdoor environments, and the combination of our indoor state estimation and general obstacle mapping and avoidance algorithms could make a vehicle system able to transition between indoor and outdoor environments by flying in and out through doorways and windows. Introducing GPS updates to the state estimate are straight forward and would allow the system to operate smoothly while transitioning between full, intermittent, and completely blocked GPS scenarios.

The offboard system also has significant room for advancement. Beyond continuing the extensions of the MIT Urban Challenge RRT planner into full 3D with an appropriate dynamic model for the quadrotor, the planner could incorporate additional computationally demanding tasks to further optimize trajectories, such as the concept of planning in information space as presented by He, Prentice, and Roy [57]. One example of augmenting the output of the DUC planner would be to adjust the vehicle heading such the the camera would look in the direction that maximizes feature persistence and the expected performance of the ego-motion estimate. This would help the the vehicle stay on track as best as possible while following the same trajectories as generated by DUC and the reactive onboard planner, which don't need to

take heading into consideration.

It would also be fruitful to provide rewards for exploration in addition to the standard penalties for obstacle proximity and excessive fuel consumption—because of the small field of view of the camera, the slice of the environment that can be mapped at any given time is relatively small, so adjusting the vehicle heading in order to form a more complete map of the environment would be very rewarding in navigating effectively. For example, the vehicle might not be able to see an opening only 45 degrees to the side and instead wanders off on a circuitous path that happens to be in front of it, rather than rotating a bit to take a peak. This would potentially decrease feature persistence which reduces estimation performance, but a "look" maneuver could be designed such that a snapshot of the tracked features before the maneuver is stored and reloaded after the vehicle performs its in-place exploration rotation, minimizing drift during the maneuver. This problem can also be addressed with a camera that has a wider field of view, but this introduces lens distortions that would need to be compensated for with additional onboard computation, and the relative advantage compared with a simple heading adjustment is unknown.

Although this advanced planning would help the vehicle work with its existing map, significant improvements can be made to the way in which it stores and thinks about the world. Techniques such as binning and connected node-graphs could greatly enhance the abilities of the offboard planner, which is currently assumed to work simply based on the point-cloud map that the onboard obstacle avoidance uses, but with less memory restrictions. In the Urban challenge, the planner was supplied a route-network of the streets and intersection to be used for high level planning. This route-network is typically not available when exploring a new building, thus the ability to develop such a network for the highest level of planning would be very advantageous. Techniques such as SIFT feature extraction and RANSAC feature mapping can help the vehicle relocate itself upon returning to a previously visited room, similar to the Monocular SLAM problem [7, 11], allowing it to localize itself within the abstract route-network, even after accumulating significant position estimate drift. All of these enhancements could be done through low-bandwidth connections to powerful ground

station computers, and could also be aided by human operators that are partially in the loop.

Finally, the capabilities of the onboard system to deal intelligently with communications failures needs to be expanded. Currently, the onboard system does nothing but wait for waypoints and land commands. If it sees something, it will made sure not to hit it, but without communication to the base station it will hover indefinitely until the battery dies and the vehicle lands, useless. A simple backtracking algorithm could be added that simply retraces the last several waypoints that were sent until communication is regained. If waypoints 1 meter apart are retained, then 1 km of backtracking could be easily stored with an insignificant 10 Kb of memory. Of course drift would grow during the retracing process, but since the vehicle can avoid obstacles in a relative sense, it would be able to make progress in the right general direction without colliding with obstacles.

6.2 Summary of Contributions

This thesis has focused on the higher level system and key supporting algorithms for the guidance and control of an HMAV in unknown indoor environments in an artificial testing situation using real sensor data on a physical vehicle. This includes the implementation and use of a hardware system that provides the sensing, computation, and vehicle for rapid prototyping, dramatically increasing the rate and productivity of development. The concepts and methodologies of the algorithms used for image processing and ego-motion estimation are discussed, and the implementations of the environmental mapping and reactive obstacle avoidance algorithms are presented. Flight results of the autonomous system are analyzed, demonstrating the viability of this solution to achieve its objectives in real environments on physical hardware. Although developed in the test bed, these technologies are directly applicable to real-world flight vehicles and many of the issues with transitioning to self-contained embedded flight systems are directly addressed with promising conclusions for the ultimate implementation of these contributions to real-world flight scenarios.

Bibliography

[1] AirRobot, Gmbh. AirRobot #01. Available at `http://www.airrobot.de/englisch/index.php`, 2008.

[2] Ascending Technologies. AscTec Hummingbird Quadrocopter. Available at `http://www.asctec.de`, 2008.

[3] Brett Bethke. Persistent vision-based search and track using multiple UAVs. Master's thesis, Massachusetts Institute of Technology, 2007.

[4] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *Robotics and Automation, IEEE Transactions on*, 7(3):278–288, Jun 1991.

[5] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 1:341–346 vol.1, 1999.

[6] Emma Brunskill, Thomas Kollar, and Nicholas Roy. Topological mapping using spectral clustering and classification. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Diego, October 2007.

[7] Zhenhe Chen, R. Rodrigo, and J. Samarabandu. Implementation of an update scheme for monocular visual slam. *Information and Automation, 2006. ICIA 2006. International Conference on*, pages 212–217, 15-17 Dec. 2006.

[8] T. Cheviron, T. Hamel, R. Mahony, and G. Baldwin. Robust nonlinear fusion of inertial and visual data for position,velocity and attitude estimation of UAV. *Robotics and Automation, 2007 IEEE International Conference on*, pages 2010–2016, 10-14 April 2007.

[9] City Comforts, the blog. Worth Noting. Available at http://citycomfortsblog.typepad.com/cities/2004/07/worth_noting.html, July 23, 2004.

[10] Javier Civera, Andrew Davison, and J. Montiel. Inverse depth to depth conversion for monocular SLAM. In *IEEE International Conference on Robotics and Automation*, April 2007.

[11] 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, June 2007.

[12] Ecole Polytechnique Fédérale de Laudanne. Flying insects and robots symposium. Monte Veritá, Asconda, Switzerland, August 12-17, 2007. Symposium website http://fir.epfl.ch, 2007.

[13] Stefan Ellis de Nagy Köves Hrabar. *Vision-Based 3D Navigation for an Autonomous Helicopter*. PhD thesis, University of Southern California, 2006.

[14] Defense Advanced Research Projects Agency. DARPA Urban Challenge. Available at http://www.darpa.mil/GRANDCHALLENGE/, 2008.

[15] Edwin Olson, Albert Huang, and David Moore. Lightweight Communications and Marshalling. Available at lcm.googlecode.com/, 2008.

[16] Alison Eele and Arthur Richards. Path-planning with avoidance using nonlinear branch-and-bound optimisation. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Hilton Head, South Carolina, August 2007.

[17] Institude for Defense and Government Enhancement. UAV summit. Herndon, VA, February 25-28, 2008. Summit website http://www.uavsummit2008.com, 2007.

[18] S.G. Fowers, Dah-Jye Lee, B.J. Tippetts, K.D. Lillywhite, A.W. Dennis, and J.K. Archibald. Vision aided stabilization and the development of a quad-rotor micro UAV. *Computational Intelligence in Robotics and Automation, 2007. CIRA 2007. International Symposium on*, pages 143–148, 20-23 June 2007.

[19] Gadget Venue. Palm-Sized R/C Helicopter. Available at http://www.gadgetvenue.com/palmsize-rc-helicopter-04274653/, 2008.

[20] Luigi Gallo, Giuseppe De Pietro, and Ivana Marra. 3D interaction with volumetric medical data: experiencing the Wiimote. In *Ambi-Sys '08: Proceedings of the 1st international conference on Ambient media and systems*, pages 1–6, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[21] William E. Green, Paul Y. Oh, Keith Sevcik, and Geoffrey Barrows. Autonomous landing for indoor flying robots using optic flow. Available at `citeseer.ist.psu.edu/green03autonomous.html`.

[22] 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. *Robotics and Automation, 2007 IEEE International Conference on*, pages 361–366, 10-14 April 2007.

[23] H. Haddad, M. Khatib, S. Lacroix, and R. Chatila. Reactive navigation in outdoor environments using potential fields. *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, 2:1232–1237 vol.2, 16-20 May 1998.

[24] Ray He. Stereo vision and mapping with unsynchronized cameras. Master's thesis, Massachusetts Institute of Technology, 2008.

[25] Gabriel M. Hoffmann, Haomiao Huang, Steven L. Waslander, and Claire J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Hilton Head, SC, August 2007. AIAA Paper Number 2007-6461.

[26] J. P. How, B. Bethke, A. Frank, D. Dale, and J. Vian. Real-time indoor autonomous vehicle test environment. *IEEE Controls Systems Magazine*, pages 51–64, April 2008.

[27] S. Hrabar, G.S. Sukhatme, P. Corke, K. Usher, and J. Roberts. Combined optic-flow and stereo-based navigation of urban canyons for a UAV. *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3309–3316, 2-6 Aug. 2005.

[28] Intel Corporation. OpenCV Computer Vision Library. Available at `http://www.intel.com/technology/computing/opencv/`, 2007.

[29] Intel Corporation. Intel Atom Processor: Intel's Smartest Chip. Available at `http://www.intel.com/technology/atom/index.htm`, 2008.

[30] Jean-Yves Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker. Available at `http://robots.stanford.edu/cs223b04/algo_tracking.pdf`, 2000.

[31] Eric N. Johnson and Daniel P. Schrage. The Georgia Tech unmanned aerial research vehicle: GTMax. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Austin, Texas, August 2003.

[32] Takeo Kanade, Omead Amidi, and Qifa Ke. Real-time and 3D vision for autonomous small and micro air vehicles. In *43rd IEEE Conference on Decision and Control (CDC 2004)*, December 2004.

[33] Jonathan Kelly, Srikanth Saripalli, and Gaurav S. Sukhatme. Combined visual and inertial navigation for an unmanned aerial vehicle. In *Proceedings of the International Conference on Field and Service Robotics*, Jul 2007.

[34] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, 2:500–505, Mar 1985.

[35] M. Kontitsis, R. D. Garcia, and K. P. Valavanis. Design, implementation and testing of a vision system for small unmanned vertical take off and landing vehicles with strict payload limitations. *J. Intell. Robotics Syst.*, 44(2):139–159, 2005.

[36] Y. Kuwata, G. Fiore, J. Teo, E. Frazzoli, and J. P. How. Motion Planning For The MIT DARPA Urban Challenge Vehicle. In *IROS (submitted)*, 2008.

[37] Y. Kuwata and J. How. Receding horizon implementation of MILP for vehicle guidance. *American Control Conference, 2005. Proceedings of the 2005*, pages 2684–2685 vol. 4, 8-10 June 2005.

[38] Yoshiaki Kuwata, Justin Teo, Sertac Karaman, Gaston Fiore, Emilio Frazzoli, and Jonathan P. How. Motion planning in complex environments using closed-loop prediction. In *AIAA Guidance, Navigation, and Control Conference*, 2008. To appear.

[39] Yves Briere Laurent Muratet, Stephane Doncieux and Jean-Arcady Meyer. A contribution to vision-based autonomous helicopter flight in urban environments. *Robotics and Autonomous Systems: Biomimetic Robotics*, pages 195–209, March 2005.

[40] John Leonard, Jonathan How, Seth Teller, Mitch Berger, Stefan Campbell, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Albert Huang, Sertac Karaman, Olivier Koch, Yoshiaki Kuwata, David Moore, Edwin Olson, Steve Peters, Justin Teo, Robert Truax, Matthew Walter, David Barrett, Alexander Epstein, Keoni Maheloni, Katy Moyer, Troy Jones, Ryan Buckley, Matthew Antone, Robert Galejs, Siddhartha Krishnamurthy, and Jonathan Williams. A perception driven autonomous urban vehicle. 2008. Submitted to *Journal of Field Robotics.*

[41] Flint Hills Solutions LLC. MAV08, 1st US-European competition and workshop on micro air vehicles. Toulouse, France, September 17-21, 2007. Symposium website `http://www.mav07.org`, 2007.

[42] Flint Hills Solutions LLC. Kansas 2009 unmanned systems symposium. Wichita, KS, April 20-21, 2009. Symposium website `http://www.fhssymposiums.com/`, 2008.

[43] L.M. Lorigo, R.A. Brooks, and W.E.L. Grimsou. Visually-guided obstacle avoidance in unstructured environments. *Intelligent Robots and Systems, 1997. IROS '97., Proceedings of the 1997 IEEE/RSJ International Conference on*, 1:373–379 vol.1, 7-11 Sep 1997.

[44] Advanced RISC Machines Ltd. ARM Cortex-A8. Available at `http://www.arm.com/products/CPUs/ARM_Cortex-A8.html`, 2008.

[45] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, pages 674–679, April 1981.

[46] R. W. Madison, G. L. Andrews, P. A. DeBitetto, S. A. Rasmussen, and M. S. Bottkol. Vision-aided navigation for small UAVs in GPS-challenged environments. In *AIAA Infotech at Aerospace Conference and Exhibit*, May 2007.

[47] A.A. Masoud. Decentralized self-organizing potential field-based control for individually motivated mobile agents in a cluttered environment: A vector-harmonic

potential field approach. *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, 37(3):372–390, May 2007.

[48] L. Matthies, A. Kelly, T. Litwin, and G. Tharp. Obstacle detection for unmanned ground vehicles: a progress report. *Intelligent Vehicles '95 Symposium., Proceedings of the*, pages 66–71, 25-26 Sep 1995.

[49] J. Montiel, Javier Civera, and Andrew Davison. Unified inverse depth parametrization for monocular SLAM. In *Robotics: Science and Systems*, August 2006.

[50] Bangalore National Aersopace Laboratories. MAV07. Agra, India, March 10-15, 2008. Symposium website `http://www.nal.res.in/mav08/`, 2008.

[51] National Oceanic and Atmospheric Administration. Fire Damages NOAA FacilityEstimated Cost $1.8 Million. Available at `http://www.noaanews.noaa.gov/stories/s676.htm`, 2008.

[52] T. Netter and N. Francheschini. A robotic aircraft that follows terrain using a neuromorphic eye. *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, 1:129–134 vol.1, 2002.

[53] Procerus Technologies. Kestrel Autopilot. Available at `http://www.procerusuav.com/productsKestrelAutopilot.php`, 2008.

[54] Arthur Richards, Yoshiaki Kuwata, and Jonathan How. Experimental demonstrations of real-time MILP control. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Austin, Texas, August 2003.

[55] E. Rimon and D.E. Koditschek. Exact robot navigation using artificial potential functions. *Robotics and Automation, IEEE Transactions on*, 8(5):501–518, Oct 1992.

[56] F. Ruffier and N. Franceschini. Visually guided micro-aerial vehicle: automatic take off, terrain following, landing and wind reaction. *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, 3:2339–2346 Vol.3, 26 April-1 May 2004.

[57] Sam Prentice Ruijie He and Nicholas Roy. Planning in information space for a quadrotor helicopter in aGPS-denied environment. In *Proceedings of the IEEE*

International Conference on Robotics and Automation (ICRA 2008), 2008. To appear.

[58] S. LaValle and J. Ku. Rapidly-exploring random trees: Progress and prospects, 2000.

[59] S. Saripalli, J. Montgomery, and G. Sukhatme. Vision-based autonomous landing of an unmanned aerial vehicle, 2002.

[60] Paul G. Savage. Strapdown inertial navigation integration algorithm design part 1: Attitude algorithms. *Journal of Guidance, Control, and Dynamics*, 1998.

[61] T. Schouwenaars, M. Valenti, E. Feron, and J. P. How. Implementation and flight test results of MILP-based UAV guidance. In *Proceedings of the IEEE Aerospace Conference*, Feb. 2005.

[62] Mark Diel Sebastian Thrun and Dirk Hähnel. Scan alignment and 3-D surface modeling with a helicopter platform. *Springer Tracts in Advanced Robotics*, 24:287–297, 2006.

[63] Jianbo Shi and Carlo Tomasi. Good features to track. In *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593 - 600, 1994.

[64] David H. Shim and Shankar Sastry. A situation-aware flight control system design using real-time model predictive control for unmanned autonomous helicopters. In *AIAA Guidance, Navigation, and Control Conference*, Keystone, CO, 2006.

[65] Tom's Hardware. Intel's core 2 quadro Kentsfield: Four cores on a rampage. Available at `http://www.tomshardware.com/reviews/cores-rampage,1316-13.html`, 2008.

[66] Tom's RC. Servo Controller SC8000 Features. Available at `http://www.tti-us.com/rc/sc8000.htm`, 2008.

[67] M.A. Turk, D.G. Morgenthaler, K.D. Gremban, and M. Marra. VITS-a vision system for autonomous land vehicle navigation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 10(3):342–361, May 1988.

[68] Berkeley Biomimetic Millisystems Lab University of California. Micro glider project. Available at `http://robotics.eecs.berkeley.edu/~ronf/MicroGlider/index.html`, 2006.

[69] Mario Valenti, Brett Bethke, Gaston Fiore, Jonathan How, and Eric Feron. Indoor multi-vehicle flight testbed for fault detection, isolation, and recovery. In *AIAA Conference on Guidance, Navigation and Control*, Keystone, CO, August, 2006.

[70] Vicon Company. Vicon Motion Capture Systems. Available at `http://www.vicon.com/`, 2007.

[71] Visual Python 3D Programming Module. Available at `http://www.vpython.org/`, 2007.

[72] R.J. Wood, S. Avadhanula, E. Steltz, M. Seeman, J. Entwistle, A. Bachrach, G. Barrows, S. Sanders, and R.S. Fearing. Design, fabrication and initial results of a 2g autonomous glider. *Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE*, pages 8 pp.–, 6-10 Nov. 2005.

[73] Allen D. Wu, Eric N. Johnson, and Alison A. Proctor. Vision-aided inertial navigation for flight control. *Journal of Aerospace Computing, Information, and Communication*, 2005.

[74] Vincent Zalzal. KFilter - free C++ extended Kalman filter library. Available at `http://kalman.sourceforge.net`, Feb. 2006.

[75] J.-C. Zufferey and D. Floreano. Fly-inspired visual steering of an ultralight indoor aircraft. *Robotics, IEEE Transactions on [see also Robotics and Automation, IEEE Transactions on]*, 22(1):137–146, Feb. 2006.