REAL-TIME IMPLEMENTATION OF VISION ALGORITHMS FOR CONTROL, STABILIZATION, AND TARGET TRACKING, FOR A HOVERING MICRO-UAV

by

Beau J. Tippetts

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

Brigham Young University

August 2008

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Beau J. Tippetts

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Dah Jye Lee, Chair
Date	James K. Archibald
Date	Clark N. Taylor

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Beau J. Tippetts in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Dah Jye Lee
Chair, Graduate Committee

Accepted for the Department

Michael J. Wirthlin
Graduate Coordinator

Accepted for the College

Alan R. Parkinson
Dean, Ira A. Fulton College of
Engineering and Technology

ABSTRACT

REAL-TIME IMPLEMENTATION OF VISION ALGORITHMS FOR CONTROL, STABILIZATION, AND TARGET TRACKING, FOR A HOVERING MICRO-UAV

Beau J. Tippetts

Department of Electrical and Computer Engineering

Master of Science

A lightweight, powerful, yet efficient quad-rotor platform was designed and constructed to obtain experimental results of completely autonomous control of a hovering micro-UAV using a complete on-board vision system. The on-board vision and control system is composed of a Helios FPGA board, an Autonomous Vehicle Toolkit daughterboard, and a Kestrel Autopilot. The resulting platform is referred to as the Helio-copter. An efficient algorithm to detect, correlate, and track features in a scene and estimate attitude information was implemented with a combination of hardware and software on the FPGA, and real-time performance was obtained. The algorithms implemented include a Harris feature detector, template matching feature correlator, RANSAC similarity-constrained homography, color segmentation, radial distortion correction, and an extended Kalman filter with a standard-deviation outlier rejection technique (SORT). This implementation was designed specifically

for use as an on-board vision solution in determining movement of small unmanned air vehicles that have size, weight, and power limitations. Experimental results show the Helio-copter capable of maintaining level, stable flight within a 6 foot by 6 foot area for over 40 seconds without human intervention.

ACKNOWLEDGMENTS

This thesis represents a large amount of time, effort, and sacrifice, cumulatively more from others than myself. I cannot adequately express the gratitude I have for the support, inspiration, and strength provided by my wife Shalae and children Savannah, Mirian, Isaac, and the one that has yet to arrive. Since the day I met her, Shalae has inspired within me a desire to lead a better life, and has helped me learn how to fulfill my potential both spiritually and secularly. My children have always been a source of strength, as I come home after long days in the trenches, they lift the heart and spirit with the squeals of excitement that dad is home. Their innocence has kept my life focused on, and increased my understanding of, an eternal perspective. I am truly grateful for an inspiring advisor, Dr. Dah-Jye Lee, whose drive to oversee and perform quality research is outweighed only by his concern for the well-being of those he works with. I have seen his work ethic and concern for his students cause himself many a sleepless night. I am also very grateful for the skill, intellect, and patience of Spencer Fowers who I was able to work with directly on this project. I owe a sincere thank you to Dr. James Archibald for his ability to encourage and inspire in every conversation. I owe a big thanks to Dr. Clark Taylor for his input and advice as well as his interest in all of my work. I want to acknowledge the contributions of many fellow students Kirt Lillywhite, Wade Fife, Barrett Edwards, Evan Andersen, Neal Johnson, Aaron Dennis, and many others who have been sounding boards and wells of insight for so many aspects of this project. I would like to also acknowledge Dr. Randy Beard and Dr. Mark Cope for their support, expertise, and excitement that definitely encouraged our progress. This work would not have been possible without any of these, all of whom I consider my friends.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation for Hovering Autonomous Micro-UAVs

As technology germinates the seeds of ideas, ideas push the development of technology to allow the ideas to be put into practice. To identify which aspect of this cycle is the origin seems as simple as doing the same for the chicken and the egg. One example of this cyclic dependency is the field of autonomous micro unmanned aerial vehicles (micro-UAVs), and more specifically hovering micro-UAVs. Of the many possible applications that have been suggested for micro-UAVs, it is difficult to know if some came before or after the emerging of the technology to apply to it. Probably of more importance than determining the origin is to realize the direction of progress, and discover and solve the problems that hinder it. It is almost certain, however, that as the technology of hovering micro-UAVs becomes more stable, both existing and new fields will find applications that are best suited for small hovering unmanned aerial vehicles.

1.1.1 Applications of UAVs

Some of the possible applications for micro-UAVs come from those suggested for large UAVs. Large hovering unmanned aerial vehicles have been proposed for use in crop dusting, remote sensing [1], cinematography, aerial mapping [2], tracking [3], inspection [4], law enforcement, surveillance [5], search and rescue, and even exploration of the planet Mars [6]. Of the possible applications for UAVs, many are not feasible due to the cost of large UAV platforms. Micro-UAVs however, are much less expensive than their larger counterparts, and so, by shrinking the platform,

the number of practical applications for UAVs expands into many more fields than previously possible.

The size of micro-UAVs are such that they can be carried by hand, transported by common automobiles, and stored in a minimal amount of space. Hovering micro-UAVs are also capable of moving about in constrained environments, such as maneuvering among buildings, trees, and poles of an urban setting, or even halls and doorways inside of a building. The ability to hover combined with their size allows micro-UAVs to get closer to objects of interest than their larger counterparts.

Of course with these advantages comes an equal share of limitations. Micro-UAVs have very small payload capabilities making it difficult to carry much computing equipment. Payload capacities are usually limited to only a few pounds and any equipment that is carried on-board must also have its own power supply, which in the case of micro-UAVs always constitutes a large percentage of unavoidable weight.

1.1.2 Tele-operation vs Autonomy

In addition to payload capacities, another limitation of many current micro-UAV implementations is that they use tele-operation [7]. Tele-operation is where one or more people control the micro-UAV remotely from a ground station. Tele-operation of a UAV requires the full attention of at least one skilled operator focused solely on controlling the UAV. This operator has to either remain within eyesight of the UAV at all times or have sufficient video feedback of the immediate area around the UAV to properly maneuver it without colliding with obstacles. At the very least, the range of the UAV is strictly limited to the effective range of the tele-operators communication system. Introducing autonomous control to the UAV can overcome many of these limitations as well as removing much of the burden of controlling the UAV from the operator, leaving the operator free to do other high-level tasks. For example, in a search and rescue application using tele-operation of a UAV, the user would have to manually control every movement of the UAV from a remote location. His attention would be focused on following the planned flight path for the UAV while avoiding collisions with obstacles. In this respect, there would be very little benefit

over a manned aircraft. If the UAV had enough autonomy to follow a flight path and avoid obstacles, then one user would be able to control multiple UAVs and have more of a monitoring role. One person could then cover much more ground with a fleet of UAVs more efficiently than other alternatives.

1.1.3 Quad-rotor vs Traditional Helicopter

There multiple options for design when choosing to develop an autonomous hovering micro-UAV, such as traditional helicopters, coaxial helicopters, and quad-rotors. The four-propeller control mechanism of a quad-rotor eliminates the complex moving mechanical parts that are required in traditional helicopters such as the swash-plate, linkage, etc. As shown in Figure 1.1, the traditional helicopter controls altitude, pitch, and roll by moving combinations of control rods connected to the swashplate up or down, which in turn move the linkage up or down and rotate the blades to the desired pitch. The coaxial helicopter also requires a more complex mechanical design to turn a hollow outer shaft the opposite direction of the inner shaft. Control of the quad-rotor is more simply a matter of ratios, specifically the ratio of power given to varying combinations of motors. For the quad-rotor, pitch and roll maneuvers are performed by transferring power from one motor to the opposing motor about the desired axis, as illustrated in Figure 1.2. The yaw angle is controlled by changing the ratio of power from two opposing motors to the other two opposing motors. Details of

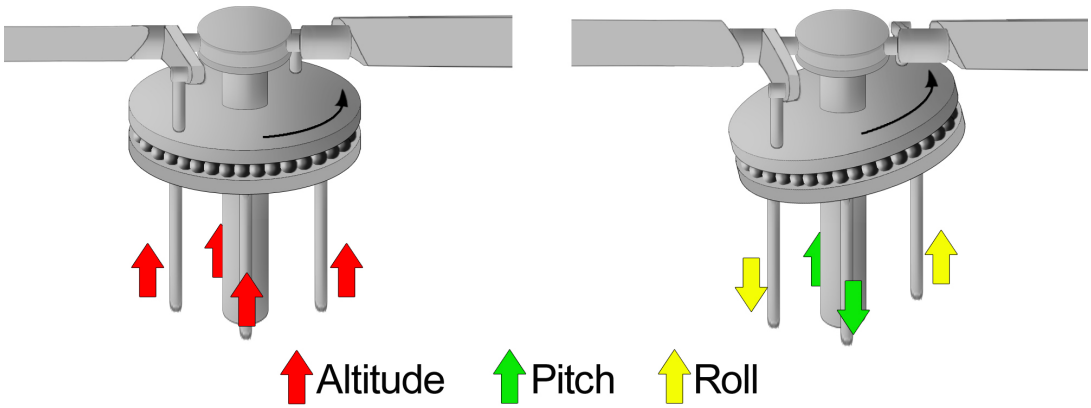


Figure 1.1: Traditional helicopter control mechanism.

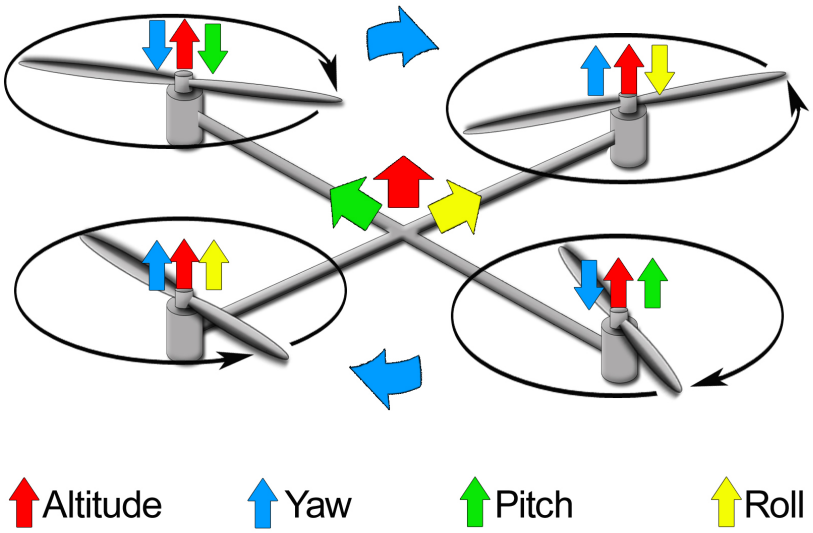


Figure 1.2: Quad-rotor helicopter control mechanism.

the physics of the quad-rotor helicopter model are discussed in [8]. Aside from being simpler in mechanical design, Bouabdallah et al. show that the quad-rotor has a better payload per volume ratio and simpler aerodynamics than a traditional helicopter [9]. These features allow for more flexibility and margin of error when constructing a custom quad-rotor platform and it does not require an advanced mechanical skill set to do so either. The quad-rotor was therefore selected as the target hovering micro-UAV platform for this work.

1.2 Problem Description

Although the quad-rotor helicopter design is a promising implementation of a hovering micro-UAV, there are issues that prevent it from being widely used in many of the suggested fields and applications. It is difficult to equip a quad-rotor with (1) adequate sensors for it to maintain stable flight and (2) sufficient computational power to process those sensors to achieve real-time control. As Ramli et al. note, on-board processing of all sensors related to control of the micro-UAV increases robustness [10]. It can be said that any vehicle that is dependent on a fixed ground station for processing of controls is not a true autonomous vehicle.

1.2.1 Low-acceleration Drift

Equipping a quad-rotor with a standard UAV inertial measurement unit (IMU) provides the ability to estimate attitude of the platform. Most IMUs are built using accelerometers and rate gyros and are required to integrate these measurements to estimate position. These sensors are also limited in their resolution, which when added to the error introduced by integrating the sensors becomes problematic for hovering UAVs. Even when the sensors of an IMU can be used to estimate angles of pitch and roll, and maintain a level hovering position, a quad-rotor can drift undetected by the IMU. Drift can be caused by slight variations in propellers and other helicopter parts, air currents in the flying environment, and other uncontrollable environment influences.

1.2.2 Vision Sensors

Ettinger et al. suggest that vision is the only practical solution to solving aerodynamic obstacles to flight for micro-UAVs, such as the drift just described [11]. Hoffman et al. also state that vision sensors would allow a quad-rotor to operate indoors [12]. A vision system can correct drift that is not detected by the attitude sensors, i.e., accelerometers and rate gyros, by tracking feature movement through consecutive images acquired by a camera facing the ground. Current solutions to correct drift and maintain stable flight using vision sensors either (1) transmit images from on-board image sensors to off-board processing platforms (e.g. [13],[14],[15]), or (2) they include off-board fixed image sensors and processors (e.g. [16],[17]). Both types of systems require images to be sent to a ground station where various image processing algorithms can be performed from which control commands can be generated and sent back to the micro-UAV. Image processing algorithms are computationally intensive and require a lot of processing power. This is the reason why most current micro-UAVs perform these computations on a ground station where large processors and high power requirements can be met [18]. The need to transmit the video and commands limits the range of the aircraft to the range of the wireless technology used. The transmission of images also introduces other problems, like

noise and delay. Noise is introduced into the images during transmission and delays occur as the UAV waits for images to be transmitted to the ground, processed, and resulting commands to be transmitted back.

1.2.3 Small Payload and Low Power

The problems introduced during transmission could be eliminated if the image processing needed to control the micro-UAV could be performed on-board. The limited payload of micro-UAVs in many cases prevents the addition of sufficient processors and power to perform anything more than simple image processing tasks on board. Specialized hardware for image processing that is small enough to be carried on-board micro-UAVs is not widely available commercially. A viable solution is required to be low-power and light-weight, and can only be considered a practical solution if it is not too disproportionate in cost to the micro-UAV platform itself. Because many image processing tasks are very computationally intensive, the on-board vision system also needs to be able to handle intense computations in real-time.

1.3 Proposed Solution

To address the issues of developing a low-power, light-weight on-board vision system, the right hardware needs to be selected. One promising hardware solution that can meet the strict requirements of an on-board vision system for micro-UAVs is Field Programmable Gate Arrays (FPGAs). FPGAs are capable of containing large digital logic circuits well-suited for image processing because similar functions can be performed repeatedly and do not require sequential processing. Many FPGAs also come with built-in processors that allow solutions to include tightly-knit hardware and software designs. This ability to parallelize computations and interact with a sequential processor make FPGAs ideal for image processing tasks. They can also require much less power than a CPU with comparable processing capabilities.

This project includes the development of an efficient quad-rotor platform as well as vision and control algorithms performed on-board that allow it to autonomously maintain a fixed hovering position and track a designated target. While

details of the vision algorithms implemented for this project will be discussed here, details of the control algorithms implemented for this project are presented by Fowers [19].

1.4 Related Work

Due to the complexity of detecting correct attitude to stabilize a quad-rotor platform, extensive research has been conducted on the topic. Efforts have been focused on topics such as enhancing sensor data for better attitude estimation and to perform specific maneuvers. Much of the work that has been done to enhance sensor data for the general scope of all micro-UAV platforms also applies to specific case of a quad-rotor as well. The following cited works are good resources to understand the problems faced in designing a stable quad-rotor and learn of some design methodologies that have created solutions.

Barrows suggests that micro-UAVs would be able to maneuver through complex urban environments if equipped with the proper vision sensor [20]. Specific examples of attempts to design such a system for quad-rotors include work done by Altüg, Earl and D'Andrea, Neff and others. Altüg et al. used a fixed ground camera to estimate position and orientation of a quad-rotor to aid in achieving stable flight [4]. Their work was later extended to use both the ground camera and a camera on-board the quad-rotor transmitting images to a second computer to achieve more autonomy [16]. Earl and D'Andrea achieved successful attitude estimation results for a quad-rotor by decomposing on-board rate gyro measurements and off-board vision sensor measurements and then applying a Kalman filter to them [17]. An actuated two-degree-of-freedom camera to help control the quad-rotor for tracking targets was successfully simulated by Neff et al. [13]. A vision-based obstacle avoidance algorithm for micro-UAVs is presented by Zufferey and Floreano [15]. A simple vision system using off-board computation hardware is proposed by Romero et al. to overcome the challenges of flying a quad-rotor indoors [21]. Chitrakaran et al. prove algebraically the case of landing a quad-rotor, assuming a monocular on-board camera to measure position information, using Lyapunov design methods [14].

All these vision-sensor-based solutions require a ground station to process the image information, making the micro-UAV dependent upon uninterrupted communication with the ground station. Vision systems have been implemented on-board large hovering UAV platforms providing feedback to allow them to land on designated landing pads [22] [23]. The size of the UAVs used in these projects were capable of carrying large computing systems.

Other types of sensors have also been used on quad-rotor platforms for attitude estimation and environment sensing. Escareno proposes a combination of an IMU with an IR sensor for indoor navigation and GPS for outdoor navigation of the quad-rotor [24]. L1 band differential GPS is used on a quad-rotor platform to overcome the problem of estimating attitude for the STARMAC project at Stanford [25], which focuses on coordination between multiple quad-rotors in areas where the necessary GPS signals are available. Castillo, Dzul, and Lozano integrated an electro-magnetic Polhemus sensor to measure attitude with a Lyapunov nested saturation controller that allowed the quad-rotor to autonomously take off, hover, and land [26]. A design methodology to build a quad-rotor is presented by Bouabdallah et al. [9]. They implement an obstacle avoidance algorithm using sonar to detect the obstacles. Many of these sensors are restricted to specific environments, like GPS units that can't recieve satellite signals in tight urban settings and indoors, and the tethered ground components of the Polhemus sensor, that could be overcome by using vision sensors.

In an effort to overcome the problem of inadequate sensing, one could implement a more robust quad-rotor platform. Patel et al. constructed a quad-rotor with a protective shroud and performed tests to ensure that it could recover from impacts with a wall and similar solid objects [6]. A neural network was implemented to increase stable control of pitch and roll of a quad-rotor by Dunfied et al. [27].

1.5 Outline

The following chapters discuss the details of the proposed solutions implemented for this project to produce a stable hovering micro-UAV platform. Each chapter begins with an introduction section that outlines what is to be presented,

followed by a background section that cites related work to support and validate the approaches discussed within that chapter. The rest of the chapter describes the details of each implementation concluding with simulated and/or experimental results of the implementation.

First, the design process of constructing the quad-rotor including experimental data used for selecting motors, propellers, batteries, etc. is described (Chapter 2). The Helios FPGA board [28] and the Kestrel Autopilot [29] used on the platform are then introduced, both of which were designed at Brigham Young University.

Details of the feature detection and feature correlation algorithms are then presented (Chapter 3). After feature information is extracted from the images the similarity-constrained homography, which includes relative changes in rotation, translation, and scale, can be calculated using the outlier-rejecting RANSAC algorithm (Chapter 4). This information is then used to control the quad-rotor allowing it to hold a steady position over a feature rich scene.

Following the description the RANSAC algorithm, the implementation of algorithms to enable the quad-rotor to change from holding a hovering position to tracking a specific object of interest are explained (Chapter 5). In order to track a target, at least two points in an image matched with their corresponding locations in a subsequent image are needed to estimate the relative movement between the two images which is measured as change in rotation, translation, and scale. A target of a minimal two colored dots was chosen to demonstrate the target tracking capabilities of the quad-rotor. To use colored dots a color segmentation algorithm was integrated with an algorithm to correct radial distortion of the camera lens, and a Kalman filter that uses a standard-deviation based outlier rejection technique (referred to as SORT) so that stable flight could be achieved (Chapter 5).

In conclusion, the direction of future work to improve attitude measurements and fine tune overall stability of the quad-rotor platform is discussed and the contributions of this work to quad-rotor research are summarized (6).

Chapter 2

Helio-copter Platform

2.1 Introduction

The first step towards implementing the FPGA-based on-board vision system on a quad-rotor micro-UAV was to acquire a suitable quad-rotor platform. Design specifications were set to require the desired platform to have a total payload capacity of 5 lbs at the current elevation of 4500 feet and achieve a flight time of 30 minutes. Various existing quad-rotor platform solutions were considered (Section 2.2). A popular commercially available quad-rotor was initially tested (Section 2.3), but had insufficient payload and flight-time capabilities. A custom platform was then designed and built (Section 2.4). The resulting quad-rotor platform consisting of mostly off-the-shelf components is able to carry a comprehensive image processing system to achieve fully autonomous flight. The on-board vision and control systems are composed of a Helios FPGA board (Section 2.5.1), a daughterboard referred to as the Autonomous Vehicle Toolkit (Section 2.5.2), and a Kestrel Autopilot (Section 2.5.3). The final custom quad-rotor platform specifications are presented along with a price breakdown of individual components (Section 2.6).

2.2 Background

2.2.1 Quad-rotor Designs

Many different quad-rotor designs have been used for research, the results of which offer valuable information when designing a quad-rotor platform. This published literature identitfies a few key platform specifications that are required to obtain a viable autonomous quad-rotor solution which include sufficient payload, power, and

computation resources. As mentioned, a truly robust autonomous quad-rotor would have all computation performed on-board the platform itself, which is limited by the payload capacity of the quad-rotor. Increasing the flight time of the quad-rotor increases its abilities, making it a more viable solution for micro-UAV applications [12]. Flight time is affected by the efficiency of motors and propellers, platform weight, and battery power. Kroo et al. discuss in detail the importance of propeller shape, dimensions, and material and the effects these have on efficiency and capability of flight [30]. This information influenced design decisions throughout the course of this project.

Some existing quad-rotor platforms include a popular commercially available quad-rotor platform called the Draganflyer which was used for research in [26], [12], [10], and [31]. A custom quad-rotor platform designed by Pounds et al., referred to as the X4 Mark II, weighs a total of 8.82 lbs. It has a payload capacity of 2.2 lbs and achieved 11 minutes of flight time with six 4-cell 2000mAh batteries. It is capable of Bluetooth communication with a range of 100 meters. Pounds et al. felt that high-frequency noise affected IMU data prevented them from flying stably untethered [8]. The OS4 quad-rotor built by Bouabdallah et al. weighs 1.14 lbs and has a payload of about 1.15 lbs. They achieved approximately 30 minutes of flight time with a 3300mAh battery [9]. None of these existing platforms ultimately met all the goals of size, weight, payload, and flight time for this project, and so consideration was given to developing a new platform.

2.2.2 FPGA-based Vision Systems

No existing platforms included an on-board vision system either. An FPGA-based solution for the vision system was considered for its strengths in image processing. Implementing image processing algorithms on an FPGA can result in much faster frame rates than software implementations, especially if software implementations require transmission to and from the micro-UAV. Using FPGAs for image processing will not only increase throughput, but also free up ground-station processors for high-level tasks, and remove the transmission-to-decision latency from

the system. If images can be processed on-board the micro-UAV, noise will not be introduced into the images during transmission, which increases image quality and results.

FPGAs have many advantages with respect to image processing and computer vision tasks. MacLean states that their ability to exploit the parallelism found in these tasks, and their ability to support different modes of operation on a single hardware substrate suggests that FPGAs are a better solution than serial CPUs and DSPs. He also explains that development time of algorithms on FPGA hardware is much shorter than on ASICs, and that although FPGAs can provide floating-point arithmetic, available resources usually require analysis of the algorithm to reduce it to fixed-point calculations [32]. Ratha and Jain show that FPGAs allow a complete real-time vision system to be implemented on a single platform [33].

2.3 Draganflyer Quad-rotor

As the focus of this work was originally intended to be the vision system a commercially available quad-rotor was first purchased from Draganflyer, shown in Figure 2.1. The Draganflyer quad-rotor came equipped with control circuitry including gyro and thermopile sensors. Its brushed motors were geared down to drive the flexible plastic counter-rotating blades. Even though the Draganflyer is small enough to fly indoors, it is dependent on its thermopile sensors for differentiation between ground and sky to maintain stability and stay level in flight. After various tests it was discovered that buildings in a typical urban environment interfered with the normal operation of the thermopile sensors. When they were not able to detect temperature differences between the ground and the sky, there was no mechanism to remove error accumulated in the gyros.

Although the inability to fly in urban and indoor environments was a major issue, the biggest problem in adopting the Draganflyer as the chosen platform was its payload capacity and flight time. Initial flights of the Draganflyer by itself without any extra equipment lasted only a couple of minutes before the motors failed due to overheating. This was due to the high altitude of BYU campus requiring higher

Figure 2.1: Commercial DraganFlyer quad-rotor.

velocities of the inefficient motors to achieve thrust levels sufficient to hover. The Draganflyer's brushed electric motors are very inefficient compared to brushless motors, and later comparisons verified that they dissipate much more energy as heat than brushless motors. The blades used by the Draganflyer were made of soft plastic which coned in flight adding to the inefficiency of the motors by losing thrust off the edges of the blades. These results required the Draganflyer to be abandoned and a new quad-rotor platform to be developed.

2.4 Building the Helio-copter

2.4.1 Mechanical Platform

Initial specifications for the custom quad-rotor included a payload capacity to support the vision system hardware for a minimum flight time of 30 minutes at an elevation of 4500 ft. Results from testing the Draganflyer indicated that these specifications would require more powerful and efficient motors, more efficient blades, and a lightweight frame that was strong enough to support itself as well as the vision system hardware.

2.4.2 Motors and Propellers

One of the major concerns with the brushed motors on the Draganflyer was that they reached extreme temperatures that caused them to fail. Initially a small brushless motor, an MTM 400F, was tested with only marginal improvement. A high-quality series of motors manufactured by Axi were eventually acquired and tested, based on recommendations from researchers in the STARMAC project [12]. Using vendor specifications, the list of possible motor sizes was reduced to four: AXI 2212/20, AXI 2212/34, AXI 2212/26, and AXI 2208/26 based on voltage and current limits. These were controlled using a Castle Creations motor controller which accepts a pulse-width modulated signal to regulate the speed of the motor. Although initial measurements indicated that four 10-amp controllers would be sufficient for our desired range of operation, these controllers generated enough heat to melt their plastic casings, and so were upgraded to Castle Creations' Phoenix model 25-amp brushless motor controllers.

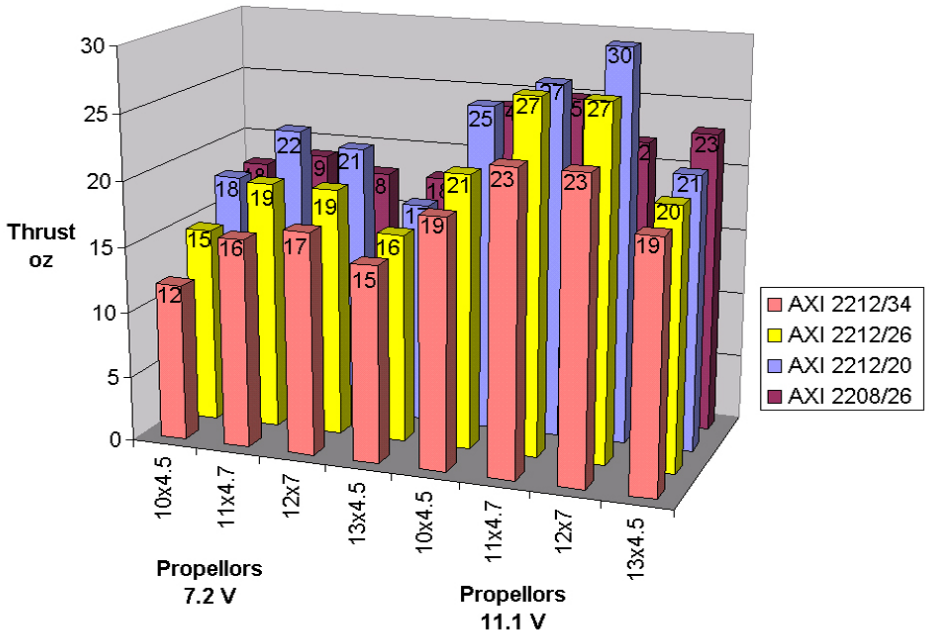


Figure 2.2: Thrust produced for combinations of motors and propellers.

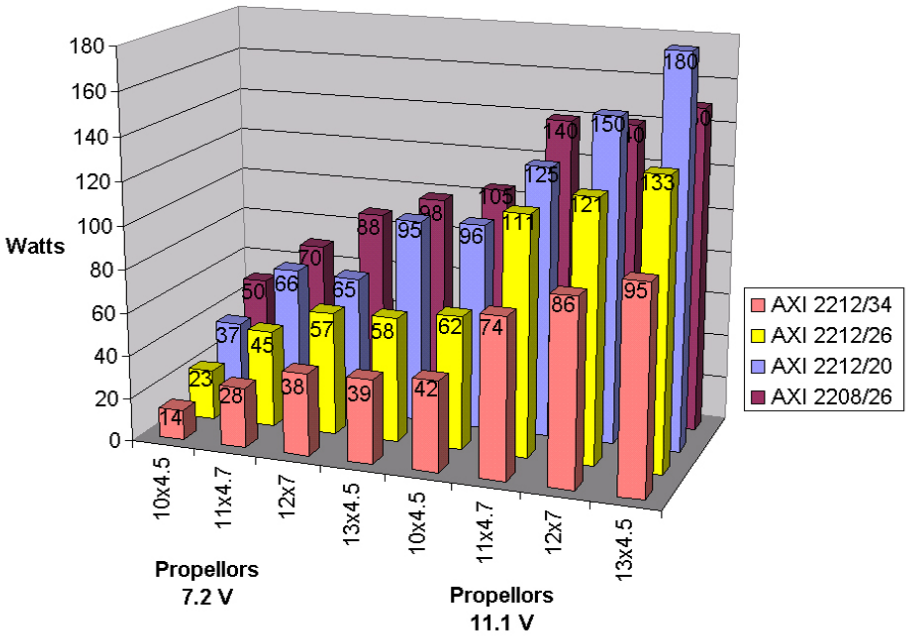


Figure 2.3: Power consumption for combinations of motors and propellers.

Due to the difficulty of manufacturing custom propellers, great effort was expended in finding a commercial propeller solution for this platform. Although there is a proliferation of model aircraft propellers, there are few that are made in matching counter rotating pairs. Of those available, all but an expensive carbon fiber propeller was purchased and tested. Figures 2.2 to 2.4 show thrust, power, and temperature results of tests run combining various motors and propellers. Each motor, represented by a different color in the graphs, was tested with each propeller, denoted along the x axis for each of two different voltage levels, 7.2 V and 11.1 V. Propellers are distinguished by their length and pitch. These tests were run for thirty seconds with the motors at maximum throttle. An RC battery watt meter was used to measure power consumption, a counter weight and scale were used to measure motor thrust, and a thermo-sensor attached to the case of the motor was used to measure motor temperature.

Projected estimates of the total quad-rotor platform weight suggested that a combined thrust of approximately 2.5 lbs would be required for the platform to hover.

To meet this requirement, the desired thrust of an individual motor at maximum throttle was chosen to be about 1.25 lbs, so that all four motors combined would produce twice the necessary thrust. This would allow the platform to hover with the motors running at midrange. As can be seen in Figure 2.2, a few motor-propeller combinations achieved this goal of close to 1.25 lbs or 20 oz. These combinations were then cross checked for power consumption and motor temperature. Comparing temperature and power consumption of motors indicates how efficient a motor is by the amount of power that is dissipated as heat and how much is converted to thrust. An inspection of Figure 2.4 suggests that the 2212/26 motor is more efficient than the 2212/34 motor because it consumes slightly more power but has lower operating temperatures. Power measurements, shown in Figure 2.3, indicate that the lower voltage battery would be a better choice while still producing close to the desired thrust with either the 11 x 4.7 or 12 x 7 propellers. However, concerns were raised about both of these propellers because the pitch of the blades is adjustable. Although this can be a desirable feature for single propeller applications like airplanes, for this

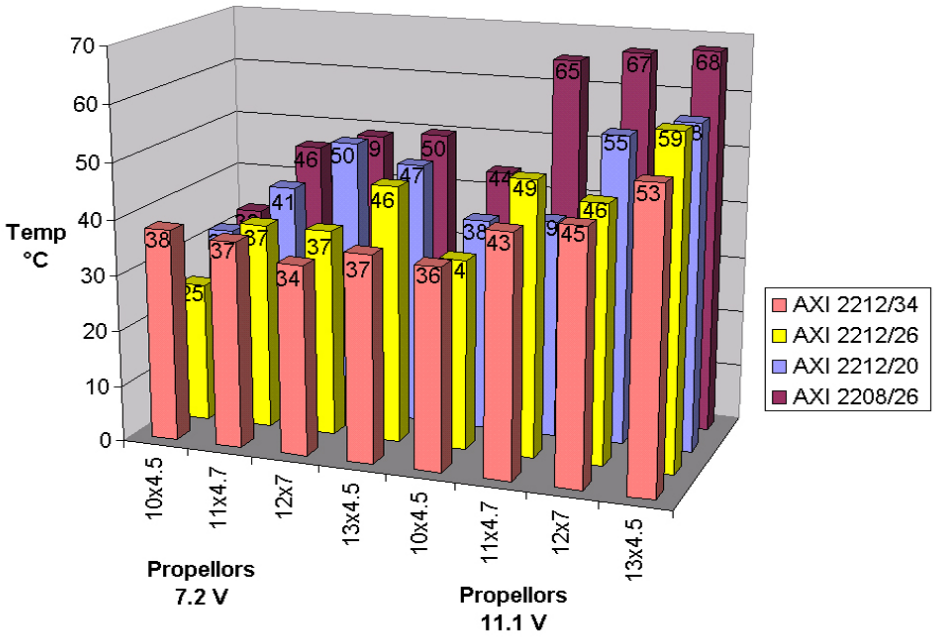


Figure 2.4: Motor temperature for combinations of motors and propellers.

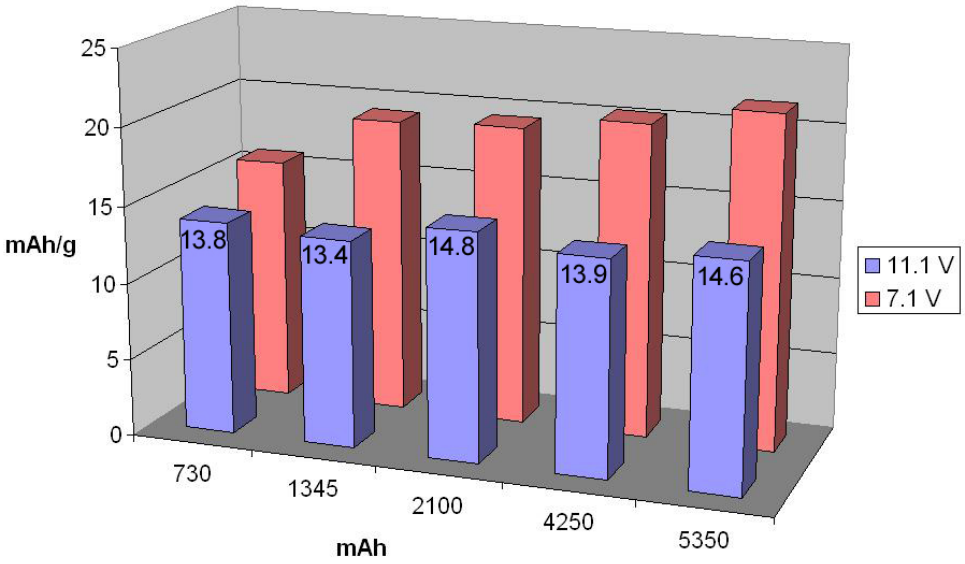


Figure 2.5: Power per weight efficiency of various sizes of lithium polymer batteries.

project it was preferred that replacement propellers have fixed, consistent behavior. This would eliminate extra time to fine-tune replacement blades so that all four quad-rotor propellers were matched. This led to the selection of the 2212/26 motor with the 10 x 4.5 propeller and an 11.1 V battery.

To select an appropriate battery, many sizes and types were compared. Lithium polymer technology has dramatically increased the power to weight ratio over other rechargeable batteries like nickel metal-hydride (NiMH) and nickel cadmium (NiCad). Different sizes of lithium polymer batteries were compared to obtain the highest power to weight ratio, while still considering the ability to incrementally trade flight time for payload capacity. As can be seen in Figure 2.5, for the 11.1 V series, the 2100 mAh battery has a slightly higher milliAmp-hour per gram rating than the other sizes.

A review of literature revealed that carbon fiber is commonly used to construct light-weight, sturdy quad-rotor frames. This is because carbon fiber composite products have a very high strength to weight ratio, and are widely available commercially. Carbon fiber rods with outside diameter of 8mm and inside diameter of 6mm were chosen for the arms of the quad-rotor. These inner diameter of the rods provided just enough room to run the three 18-gauge motor wires to the controllers which were

Figure 2.6: Motor mount, motor, and propeller.

placed at the center of the frame. A simple nylon motor mount was machined to secure the motor to the end of the rod. The completed motor assembly is shown in Figure 2.6.

The carbon fiber rods were connected at the center of the quad-rotor with a nylon block. This block also provided a surface to which the electronic equipment could be attached. A rapid-prototyped ABS plastic frame was developed, consisting of two oval rings, which was attached to the rods. The vision and control hardware were

Figure 2.7: Helio-copter platform.

then mounted to this frame, which provides protection to the equipment. The frame allowed the batteries to be placed towards the bottom of the quad-rotor, effectively lowering the center of gravity and increasing the stability of the platform. Figure 2.7 shows the completed Helio-copter platform.

2.5 Computing Hardware

Completion of the simple yet sturdy quad-rotor frame allowed electronic hardware for sensing and control to be integrated with relative ease. The block diagram in Figure 2.8 shows a system overview of the quad-rotor platform with the on-board vision and control systems. It consists of the Helios FPGA board coupled with the Autonomous Vehicle Toolkit daughterboard (AVT) which communicates with the Kestrel Autopilot (KAP) through a serial connection. The KAP controls the quad-rotor motors by pulse-width modulated signals to the electronic speed controllers (ESC). The CMOS image sensor connects directly to the AVT board where the im-

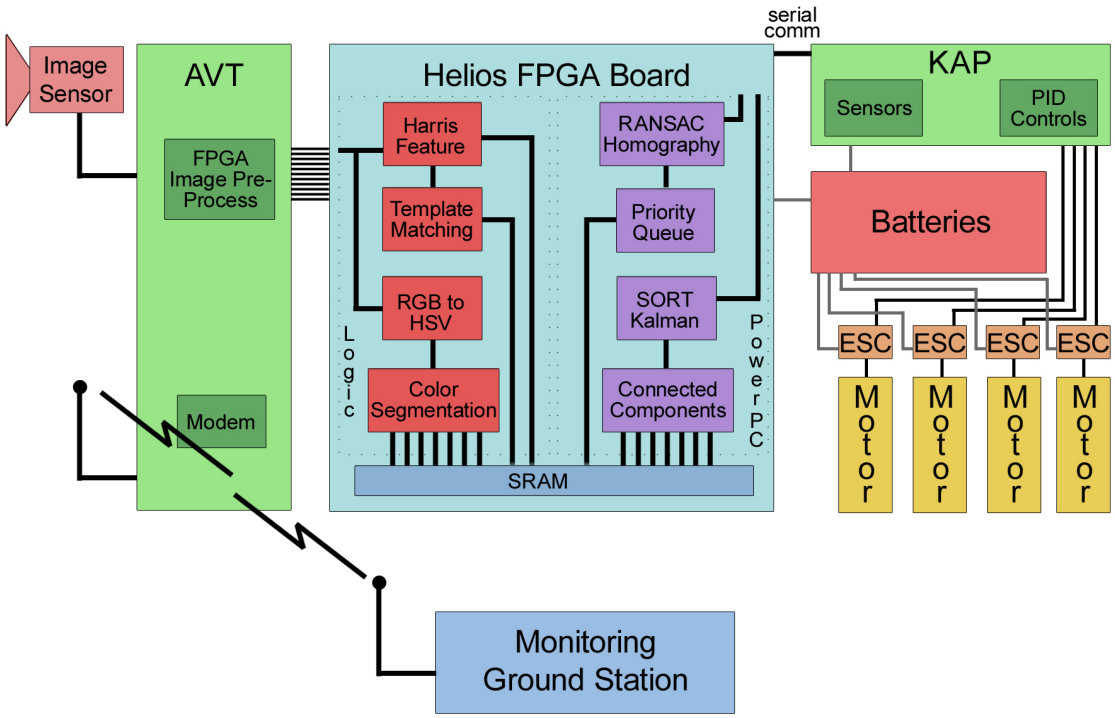


Figure 2.8: Overview diagram of the Helio-copter on-board vision and control system.

age is preprocessed by a Spartan FPGA before being relayed to Helios for full image processing.

2.5.1 Helios FPGA Board

In order to do on-board image processing on small autonomous vehicles, a compact, low power FPGA board called Helios (shown in Figure 2.9) has been developed by the Robot Vision Lab at BYU [28]. It is approximately the size of a deck of cards and in typical configurations consumes 1 to 3 watts of power, making it ideal for various micro-UAV applications. The Helios board has on-board memory in the form of SDRAM and SRAM. It has serial and USB connections. The current version of Helios contains a Virtex-4 FX60 FPGA that is equipped with an embedded PowerPC 405 module and over 56000 configurable logic blocks. The PowerPC 405 runs at a maximum of 450 MHz, and the configurable hardware can run at up to 500 MHz. The Helios also has a 120 pin header that allows it to connect to daughterboards built to include less general, more application specific hardware. For example, for its first application Helios was used on a small ground vehicle, and was equipped with the Ground Vehicle Board (GVB) that included a wireless modem, a camera header for a Micron MT9V111 CMOS sensor, and several I/O pins to control servos and read encoders.

Figure 2.9: Helios FPGA board designed at Brigham Young University (actual size).

Helios has been used for a variety of vision processing tasks at up to 60 frames per second, providing real-time vision processing on a small, low-power, light-weight, portable platform. As an example of its capacity and ability, it was configured to simultaneously compute and store ten sequential 640 x 480 images of each of the following: an original RGB image, the HSV conversion of the original, 8 color-segmented images, a rank transform image, a Harris feature image, and template matched feature correspondences between each pair of Harris feature images. The Helios processed all these achieving frame rates over 37 frames per second and occupied only about 1/3 of its logic blocks.

As seen, the Helios FPGA board offers lots of processing power, and using it on the quad-rotor avoids transmission issues such as noise and time delays, allowing it to obtain higher quality images in real time and maintaining controlled flight without a wireless tether. The resulting on-board vision system allows autonomous control algorithms such as drift control and target tracking to be implemented directly on the quad-rotor itself. The integration of the Helios FPGA board as the vision system for the quad-rotor has created a unique hovering micro-UAV platform, and has been dubbed "Helio-copter".

2.5.2 Autonomous Vehicle Toolkit

The Autonomous Vehicle Toolkit (AVT) daughterboard was designed specifically for untethered autonomous applications. It has an MMC flash memory unit which can be used for extra data storage, and ZigBee wireless modem headers for wireless communication of up to one mile. The AVT board includes a video DAC and an analog video transmitter. This allows the transmission of a video stream wirelessly to a TV, or computer with a framegrabber, for real-time visual feedback. The AVT also includes general-purpose I/O pins, and connections for up to two CMOS image sensors. The circuitry on the AVT for the CMOS image sensors has been designed to allow them to link to each other to enable a stereoscopic mode. The AVT board is also equipped with a Spartan-III FPGA meant to preprocess images from the two image sensors, after which it relays the images to the Virtex4 FPGA on Helios.

Table 2.1: Image preprocessing tasks resource usage of Spartan FPGA.

	AVT Spartan		
	Used	Capacity	%
Slice Flip Flops	1,463	7,168	20%
4 input LUTs	979	7,168	13%
Slices	1,028	3,584	28%
Bonded IOBs	37	97	38%
RAMB16 / BRAM	3	16	18%

To improve real-time results the Micron MT9V111 image sensors were upgraded to Micron MT9V022 Global-SNAP CMOS sensors. The MT9V111, like most CMOS image sensors, has a rolling shutter where cells are exposed one row at a time from top to bottom, pipelining out the image data serially as it goes. This becomes a problem when there are quick movements in the scene, because the scene shifts as each row of the sensor is exposed, resulting in frame shear. The Global-SNAP camera exposes the whole sensor at one time and then buffers the information so that it can transmit it while the next frame is exposed. The MT9V111 sensor comes in an SOC (sensor-on-a-chip) package, which outputs pixel data in many image standards, RGB565, YUV 4:2:2, etc. The MT9V022 is not available in an SOC package and therefore required preprocessing of its raw Bayer RGB output. This preprocessing

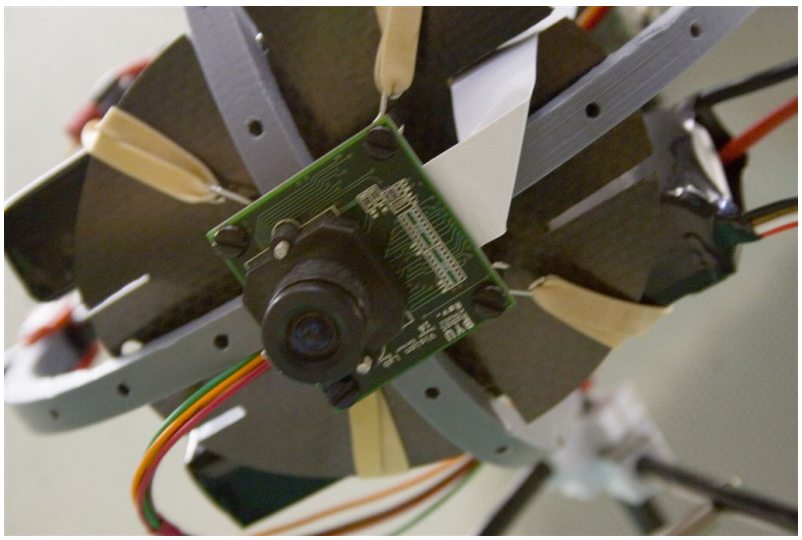


Figure 2.10: Micron MT9V022 CMOS image sensor.

was implemented on the Spartan-III FPGA which included demosaicing the Bayer RGB image, color balancing the image, and formatting the data into the RGB565 format. The hardware to implement these tasks used only a small percentage of the resources on the Spartan-III, as seen in Table 2.1. The mounted image sensor is shown in Figure 2.10. It has been mounted facing downward, towards the ground, to provide a sensor measuring platform movement relative to the ground.

2.5.3 Kestrel Autopilot

The Kestrel Autopilot (KAP) seen in Figure 2.11 was designed in the Magicc Lab at BYU for autonomous control of micro-UAVs [29]. It consists of an inertial measurement unit (IMU) integrated with a 29 MHz Rabbit processor. It has 3-axis angular rate and acceleration measurement, barometric sensors for altitude measurements and a magnetometer to measure changes in yaw. The KAP also incorporates temperature compensation to increase accuracy of all the sensors. It can communicate through its four serial ports and four servo ports. It is a very small, lightweight system that can control the Helio-copter without sacrificing a lot of weight for complicated sensing equipment. The Magicc Lab also developed ground station software called Virtual Cockpit that provides wireless communication and system monitoring of the KAP.

Figure 2.11: Kestrel autopilot designed at Brigham Young University.

The altitude sensors on the IMU are ineffective indoors. Air conditioning systems inside most building regulate temperature and pressure which makes the readings from the barometric sensors invalid. Altitude measurements are provided by the vision system for this project.

For this work, the KAP was programmed with PID control loops to command the quad-rotor motors. It was configured to accept desired values for pitch, roll, yaw, and altitude from the Helios board through a serial connection which are compared to attitude measurements from the IMU to generate error terms for the PID control loops. In this manner the vision algorithms implemented on the Helios could compute desired attitude corrections and pass them on to the KAP. Details of the control design and algorithms for the Helio-copter are described by Fowers in [19].

2.5.4 Platform Communication Development

To be able to test the Helio-copter vision and control systems, an existing software application was adopted called Virtual Cockpit. Virtual Cockpit is a GUI that was designed for mission-level control and monitoring of the Kestrel Autopilot. The software allows the user to remotely monitor all sensor readings and KAP state through wireless communication. The software also provides a logging and debugging environment for platform testing. Virtual Cockpit has addressing features that can be used when multiple autopilots are available to communicate with. By configuring an unused address and assigning it to Helios, logging and debugging functionality provided by Virtual Cockpit could be used for Helios as well. This also simplified full platform testing phases by provided one debugging environment for both the Helios and the KAP. Eventually, integration efforts will include implementing the Virtual Cockpit high-level mission commands on the Helios, and extending Virtual Cockpit to include a display tab for analog video feedback from the Helios board.

2.6 Results

The Helio-copter meets the design specifications that were detailed at the outset of the design process. It is capable of carrying all necessary vision and control

electronics, two batteries that last for close to 20 minutes of flight, as well as motor capabilities of almost an extra two pounds of payload. If extra payload is used for more batteries, the Helio-copter should be able to obtain flight times in excess of 30 minutes, even at the current elevation of 4500 ft. A breakdown of weight by individual components is given in Table 2.2. An overview of the physical specifications of the completed quad-rotor platform are given in 2.3. Table 2.4 shows a price breakdown of the complete system.

Table 2.2: Heliocopter weight breakdown by component.

Component	Quantity	Unit Weight (oz)	Total Weight (oz)
AXI 2212/26	4	2.01	8.04
Phoenix-25 ESC	4	.63	2.52
Prop 10x4.5	4	.22	.88
Frame	1	9.75	9.75
Helios/AVT	1	3.43	3.43
KAP	1	1.53	1.53
MT9V022 Image Sensor	1	.37	.37
Lipo 2100mAh 3S	2	5.29	10.58
Total			37.1

Table 2.3: Helio-copter physical specifications.

Specification	Actual	Spec
Total Height	9 in	*
Total Width	30 in	*
Total Length	30 in	*
Thrust	1.22 lbs/motor	1.25 lbs/motor
Total Weight	2.32 lbs	2.5 lbs
Approx. Flight Time	20 min	30 min

(* no design specifications were given other than to be as small as possible)

Table 2.4: Helio-copter electrical and mechanical specifications.

Item	Qty	Price
AXIS brushless motors	4	$75/ea
Castle Creations 25A speed controllers	4	$75/ea
Carbon fiber tubing, acrylic, nylon, and ABS plastic frame	N/A	$150
Helios FPGA board	1	$5,000
* Xilinx Virtex4 FX-60 FPGA		
* USB 2.0		
* 64MB SRAM		
* 8MB SDRAM		
* 16MB Flash RAM		
AVT Daughterboard	1	$200
* Spartan-III FPGA		
* GPIO Headers, 3.3V and 5.0V		
* 5.0V switching regulator for PWMs		
* ADV7171 DAC for NTSC image transmission		
* Maxstream XBee Zigbee wireless header		
* Micron MT9V022 CMOS Global-SNAP image sensor	2	
Kestrel Autopilot	1	$5,000
Lithium-polymer (LiPo) 20C 11.1V 2100mAh batteries	2-3	$80/ea

Chapter 3

Feature Selection and Correlation

3.1 Introduction

In order to provide image processing solutions on-board micro-UAVs, a balanced combination of software algorithms and hardware implementations needs to be designed. This work describes an on-board image processing solution to detect and track features that are used to estimate the movement or homography from one frame to the next for the Helio-copter.

Feature detection forms the basis of many UAV applications. Features are fairly unique identifiable points in an image. Detected features can be used for many algorithms, ranging from obstacle avoidance to target tracking. Tracked features may also be combined with line detection or color segmentation in order to implement path or lane following. Features can also be used to obtain information about a UAVs surroundings, such as height above ground, pitch, roll, and velocity. The right features in an image can even give an accurate representation of motion in the frame. This project includes algorithms to detect features in images and correlate individual features from one image to the next. These algorithms are implemented using hardware/software co-design to achieve real-time attitude and motion information on the Helio-copter. Hardware implementation of other vision algorithms, such as an optical flow algorithm, is also being researched in the Robotic Vision Lab at BYU [34].

The following sections will first cite various works to support the choice of the Harris feature detection algorithm ([35],[36]) as the feature detection algorithm for this implementation (Section 3.2.1). Then the effectiveness of template matching to correlate features in sequential images will be discussed (Section 3.2.2). In the process of implementing these two algorithms, a simplified approach to correlate features using

the minimum distance and priority queue was implemented as well. The details of how these algorithms were implemented in hardware are then presented (Sections 3.3 and 3.4). Finally both simulation and experimental results of the algorithms are discussed (Section 3.5).

3.2 Background

Image processing solutions involving feature detection currently exist for micro-UAV applications, but all are performed remotely on a ground station computer and therefore are limited by the noise and latency issues mentioned [16],[13],[21],[11].

3.2.1 Harris Feature Detector

Different feature detection algorithms possess varying strengths and weaknesses; therefore, a closer look is required to determine the appropriate algorithm for this implementation.

As a micro-UAV moves across a scene, its motion can be estimated using an affine model [37]. Therefore, it is important that the selected feature detector can find the same features in a scene as the scene is translated, rotated, or scaled with reference to the UAV. Although an on-board image processing solution has considerably less noise than a majority of current micro-UAV solutions which transmit video and process information on the ground, the feature detector also needs to be robust to noise that is experienced on micro-UAV platforms. Schmid, Mohr, and Bauckhage found that an improved Harris feature detection algorithm had a better repeatability rate than the Foerstner, Cottier, Heitger, and Horaud algorithms when images were modified by rotation, warping, scaling, lighting, etc.[38] The results of their work show that the Harris feature detector algorithm is robust even when these changes to the images are severe, except in the case of scaling. In [37], Johansen discusses feature detection algorithms that are better for micro-UAV applications. He found that Harris feature detection worked better than Canny Edge detection, Foerstner interest operator, Binary Corner detector, and gradient differencing using a Sobel kernel. In another study of feature detection algorithms, Tissainayagama and Suter

also found that Harris feature and KLT feature tracking algorithms performed better than Kitchen-Roselfield and Smith corner detection both in general performance measurements and performance under noisy conditions [39].

3.2.2 Template Matching

Johansen also compares the template matching to other popular feature tracking algorithms like profile matching and optical flow. His results show that template matching performed better for micro-UAV applications [37]. He discusses how a simplifying pyramidal technique is needed to increase the speed of template matching to meet real-time performance when implemented in software, but that this also results in less accurate tracking. The German Aerospace Center, Institute of Flight Mechanics implemented template matching on a full-size helicopter platform for hover stabilization. The size of the platform allowed multiple processors to be carried onboard. This allowed the image processing to maintain a frame rate of roughly 30fps [40]. Maintaining accuracy and high frame rates without multiple processors becomes possible by implementing the algorithms in hardware. Another hardware implementation of template matching was performed by Toyota Central R&D Labs. Researchers used a "Motion Estimator Processor" to implement template matching on ground vehicles. This system was used to allow one vehicle to track the position of another vehicle and follow it [41].

The aforementioned articles implemented these algorithms either wholly in software, or in commercially-available image-processing hardware. Researchers at the Imperial College, London implemented template matching on an FPGA. Using a Virtex 1000E FPGA, they were able to process HDTV quality video 7,000 times faster than a 1.5Ghz Pentium 4 computer [42].

3.3 Harris Feature Detector Implementation

After selecting the Harris feature detection algorithm, it was organized into hardware components. An system overview of these components is shown in Figure 3.1. Implementing the Harris feature detector in hardware required the use of buffers

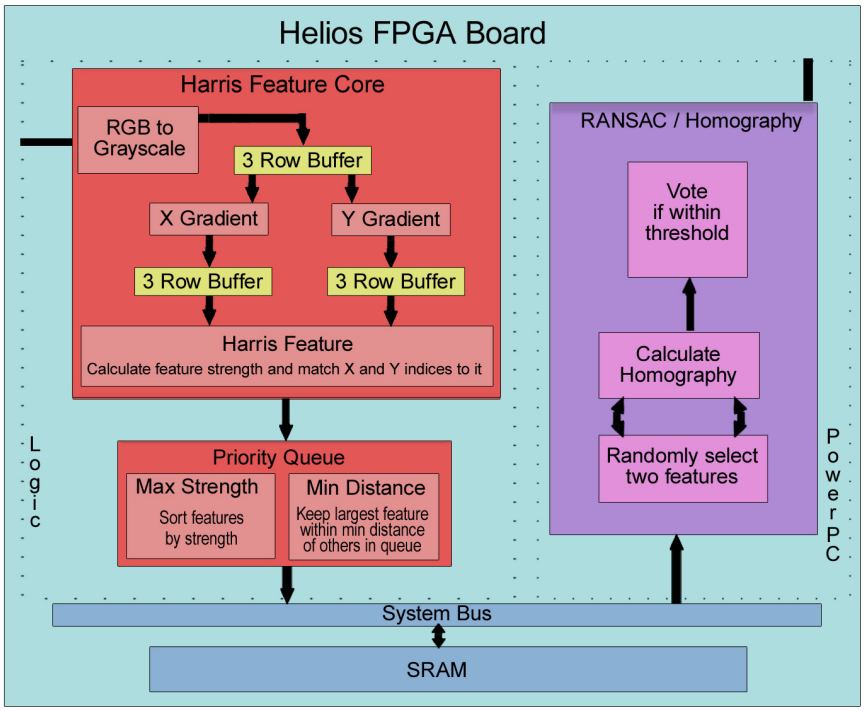


Figure 3.1: Block diagram of Harris feature core and priority queue core.

that could accumulate three rows of grayscale image data, x and y gradient blocks, and a Harris feature block. The x and y gradient blocks are both fed by a 3×3 pixel matrix accumulated by a row buffer, shown in Figure (3.1) as the top-most yellow block. The blocks were designed to allow pipelined processing of images. The gradient cores each output one single-pixel gradient value for each 3 pixel by 3 pixel matrix input to them. The x gradient (I_x) value is found using the kernel in Equation (3.1). The y gradient (I_y) is found using the kernel in Equation (3.2). The gradient values are then buffered for three rows in preparation for the Harris feature detection core, represented by the lower yellow block in Figure (3.1). The Harris core calculates feature strengths and matches strengths with x and y indices representing the pixel location in the image. The feature strength, x index, and y index are concatenated to create a single 32-bit value.

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \tag{3.1}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}. \tag{3.2}$$

Equations 3.3, 3.4, and 3.5 outline the Harris Corner detection algorithm:

$$\begin{aligned} I_x(x,y) = I(x+1,y-1) - I(x-1,y-1) + I(x+1,y) - \\ I(x-1,y) + I(x+1,y+1) - I(x-1,y+1), \end{aligned} \tag{3.3}$$

$$\begin{aligned} I_y(x,y) = I(x-1,y-1) - I(x-1,y+1) + I(x,y-1) - \\ I(x,y+1) + I(x+1,y-1) - I(x+1,y+1), \end{aligned} \tag{3.4}$$

$$C(G) = det(G) + k * trace^2(G), \tag{3.5}$$

where

$$G = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}. \tag{3.6}$$

After finding the image gradient, G and $C(G)$ can be computed. $C(G)$ is the value that represents the "cornerness" of each pixel.

In order to perform Harris feature detection in hardware a few modifications were made to the original algorithm. While Equation (3.3) and (3.4) is evaluated three signed numbers are produced when pixel data $I(x,y)$ is an unsigned eight bit value. The sum of these three numbers will require more than eight bits to avoid overflow issues. Since more bits are already required and only the magnitude of each

of the three intermediate values is important, some FPGA DSP multipliers were used to square these values to remove the sign. The three resulting 16 bit values are then shifted down four bits and summed together. All numbers are now represented by 12 bits and can be multiplied according to Equation (3.6). This allows the determinant of G to fit in 24 bits and the trace of G to fit in 26 bits. A value of 0.04 was used for k, and the result $C(G)$ was truncated to 13 bits so that when coupled with the x and y indices created a 32 bit value. This truncation introduced an automatic threshold of feature strengths.

3.4 Feature Correlation

3.4.1 Priority Queue

Initially the Harris feature core was coupled with a priority queue hardware block. The feature priority queue operates on the 32-bit values output from the Harris feature block, prioritizing them based on feature strength. It uses a queue that contains a sorted list of twenty features. When the processor receives the interrupt signifying that the frame is complete, then it empties the queue, writing the values to memory. Sorted lists from two sequential images are then compared in software, denoted by the purple blocks in Figure 3.1, and features are matched when they are found within the minimum distance in pixels of each other. The priority queue core can be seen below the Harris feature core in Figure 3.1, which shows the flow of data through the system.

As each feature strength and pair of indices are produced by the Harris feature core, it is compared with existing pairs in the queue. The new pair is inserted in the appropriate location in the queue only if there are no pairs with a higher feature strength already in the queue that are within a specified minimum distance of the new pair. This operation in the priority queue is accomplished in hardware by making two passes through the pairs in the queue. The first pass compares the x, y indices of the new pair with the x, y indices of the pairs in the queue using a Manhattan distance in terms of pixels. When a pair is encountered that is within the minimum distance then the pair with the lower strength value is set to all zeros. The second

pass compares feature strengths allowing the new feature strength index pair to be inserted into the queue if necessary and the end of the queue to shift down. The resulting prioritized feature list produced by this pipelined implementation does vary slightly from what a full sort of all features in the image would produce. Suppose the case where three of the top twenty features are located close enough to each other that each pixel is within the minimum distance of its adjoining features, but that the first and third feature are not within the minimum distance of each other. If the third feature was greater than the second, and the second feature was greater than the first, then first would be removed from the queue by the second feature, and the second feature would be removed by the third. In the case of a full sort, the first feature would still be found in the queue, but in this implementation it is not returned to the queue.

Limiting the selection of features using a minimum distance minimizes the grouping of features and provides a simple matching scheme across sequential images. Using features that are spread out across the image decreases the probability of a moving object in the scene to negatively affect the homography that is calculated. If a minimum distance threshold is selected that corresponds to the maximum movement a feature is expected to have from one frame to the next, then the matching scheme is as simple as correlating the two features that are within the minimum distance of each other. A minimum distance threshold was found empirically by flying the Helio-copter at a maximum horizontal velocity, and logging a sequence of images that could be analyzed.

Although feature correlation results using the hardware priority queue were promising (Section 3.5.2), a more robust template matching solution that could handle correlating more than twenty features was also used on the Helio-copter.

3.4.2 Template Matching

An existing template matching hardware core developed in the Robot Vision Lab was also used on the Helio-copter. The template matching feature correlator core uses a template of 8×8 pixels around each feature given by the Harris feature detector

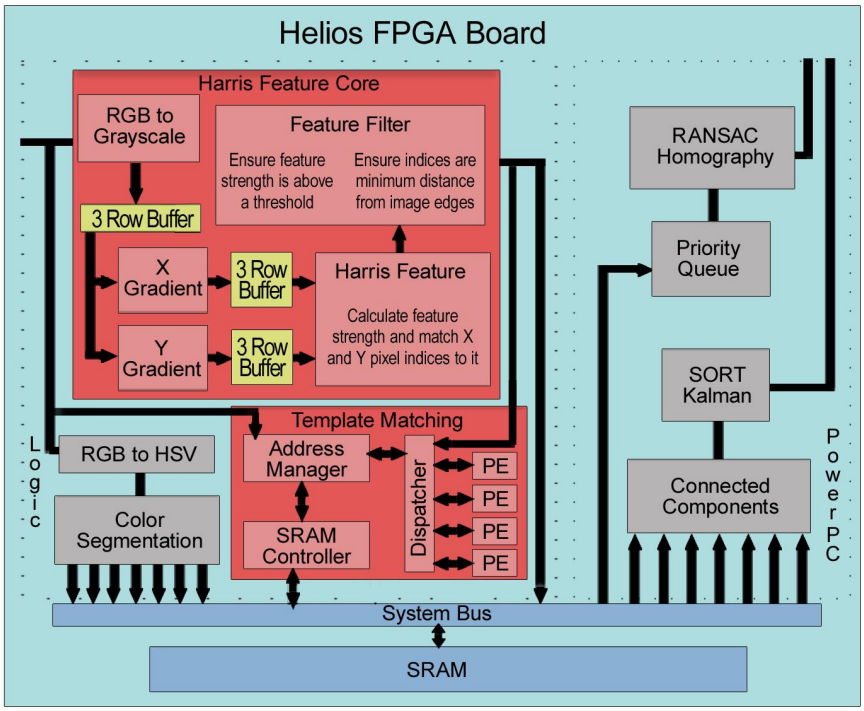


Figure 3.2: Block diagram of Harris feature core and template matching core.

core to search for the same feature within a 32×64 window around the same index in the subsequent image. This core is made up of four different types of hardware blocks, one block to manage image addresses in memory, a block to interface to SRAM, blocks to calculate Sum of Squared Differences (SSD) between the template and the search area called processing elements, and a block to dispatch templates and search areas to the processing elements. Once the template matched features are correlated then the list of paired feature indices are stored in SRAM and an interrupt is asserted to signal the RANSAC homography software to start processing the data. A diagram showing how each of the blocks are connected this version of the hardware is shown in Figure 3.2.

Image Address Manager

The template matching correlator core receives the current grayscale image and stores it in SRAM. The 4MB SRAM that is available on the Helios board was

used to store up to eight images for this core. The image address manager block in this core rotates through the SRAM keeping track of the address of the current image and the previous image. It also tracks the status of the current image, whether or not it has been completely written, so that it can signal the template matching process to start.

The address manager also uses a specific addressing convention to allow simple image data accesses in SRAM. To accomodate the camera's maximum image resolution of 640 x 480 pixels, the address used by the address manager is composed of ten bits for the y pixel index and eight bits for the x pixel index, as well as three bits to distinguish between up to eight images in memory. This addressing scheme allows quick access to individual pixel data needed for the template area from the previous image and the search area from the current image.

SRAM Controller

A dual-port SRAM Controller was implemented to handle image data transfer to and from the external SRAM on Helios. The controller took advantage of a three-stage pipeline to hide the three-cycle latency of the SRAM, so that read or write requests could be issued on every clock cycle. This allowed the full bandwidth of the SRAM to be available, minimizing the affect that the bottleneck of memory accesses has on the system.

Dispatcher

The dispatcher block receives feature x,y indices from the Harris feature block and sends a request to the address manager for a template area from one image and a search area from the subsequent image, and then assigns a processing element to compare the areas. The processing elements require a set amount of time to complete execution and do not lend themselves to a pipelined architecture. This causes the dispatcher to have idle periods waiting for a processing element to become available. Feature locations from the Harris feature block are buffered by the dispatcher, al-

lowing it to keep all the processing elements fully utilized without losing incoming features.

As is the case with most algorithms, implementing template matching in hardware requires modifications to the algorithm to maximize efficient use of FPGA resources. One modification that was made concerned the size of the template used to estimate how much a feature matches. Standard approaches use a symmetric number of pixels on all sides of a feature location, which results in odd number sized templates, e.g. 5×5, 9×9. FPGA storage resources required to implement a template matching algorithm, like Block RAMs, have even, multiple-of-two sized ports. To maximize efficient use of the read and write operations for these resources an asymmetric 8×8 pixel template is used instead of the standard symmetric templates. The four-byte ports on the Virtex-4 FPGA BRAMs allow a full template to be written to them in 16 operations.

The SRAM has the ability to burst four bytes in one read operation if the starting address is aligned on a four-byte boundary. Reading the template area from SRAM could be completed in as few as 16 read operations. If the template were not aligned, it would require at least 24 read operations, three operations per row of the template, as well as requiring control logic to extract the correct eight bytes from the twelve read in. Instead of adding this hardware, the dispatcher was designed to request the template area one byte at a time, increasing the latency to a maximum 64 read operations. When contrasted with the 512 read operations required for the search area, the extra 40 read operations for the template become less significant than the area saved by not implementing control logic.

The size of the search area that the template is compared against was also constrained by the efficient use of FPGA resources. Traditionally the trade-off concerning the size of the search area is CPU processing time versus the probability that the feature has not moved outside the search area by the next frame. Because so many search areas can be processed in parallel on the FPGA, the concern shifts away from reducing processing time to being able to fit the search area into the local BRAM storage. BRAM sizes on the Helios FPGA are 2048 bytes which allowed for a search

window of 32×64. A square search area is preferred to a rectangular one because it allows for uniform searching in all directions from the original feature location, but once again this was sacrificed to conserve FPGA resources.

To overcome the same four-byte boundary issues for the search area that were encountered with the template area, the search area was also implemented with a variable asymmetric radius. This was done by truncating the bottom two bits from the starting address of the search area, so that it always falls on a four-byte boundary. This was done only to the part of the address that corresponds to the x index, since the layout of memory does not cause the same problem over multiple rows.

The dispatcher has separate address registers for the template and search areas with the ability for the processor to set these registers. This functionality provides the ability for a feature prediction algorithm to be implemented in software to center the search area at a different location than the original feature location. If problems arise in a given application due to the limited search area, this allows for a solution to overcome the problem.

Processing Elements

The processing element (PE) blocks perform all the template matching calculations. Multiple processing elements are instantiated each containing a BRAM for template and search area storage. Each element calculates a sum of squared differences (SSD) between the 8 x 8 template and 8 x 8 windows in the search area. The 8 x 8 window in the search area that has the smallest SSD with the template is considered a match and the center x and y indices are stored with the original features x and y indices as a feature correlated pair.

The main bottleneck of the template matching process is the memory access bandwidth of the FPGA BRAMs. There are two read/write ports for each BRAM that are each four-bytes wide. This means a maximum of eight bytes can be read per clock cycle, and a whole row of the template can be calculated in parallel. To take advantage of this, four processing elements are each given a byte-shifted copy of the

same search area, allowing a full search area for one feature to be completed in about 3000 clock cycles or 30μs.

3.5 Results

3.5.1 Harris Feature Detector

After completing the implementation of the Harris feature detector hardware, multiple tests were performed to ascertain performance results. Figures 3.3(a) and 3.3(b) show a typical image scene where boxes representing the minimum distance defined in the priority queue are drawn around the resulting features on the image. Figure 3.3(a) is the original image and Figure 3.3(b) is the top eight bits of the Harris feature strengths of the image.

Comparing the sample scene and the resulting feature strength image generated by the Harris feature detection core shows that feature strengths are calculated and located where they would be expected. Given the design of this implementation it would be possible to track more features, the only limitation being the time to sort them in the priority queue.

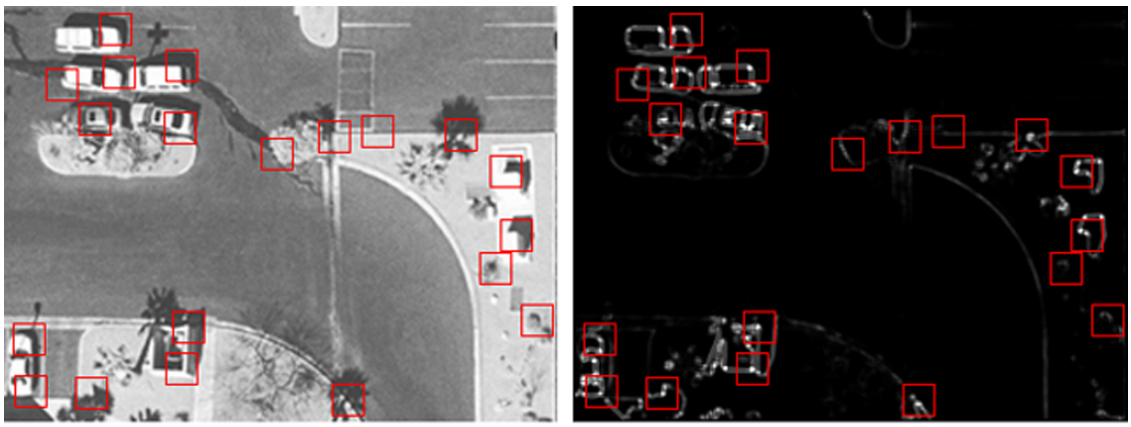


Figure 3.3: Sample images from Harris feature core. (a) Typical scene for micro-UAVs with minimum distance boxes centered on features. (b) Feature strengths of same scene.

3.5.2 Priority Queue

The minimum distance correspondence scheme gave excellent results. The priority queue was able to correspond the 20 strongest features at 30 fps. It was found that features were mostly mismatched in just one case where two features that were within the selected minimum distance of each other in a scene produced very similar Harris feature strengths. Sometimes these features could vary in strength enough to switch their order, changing which one stayed in the queue. If one is selected in the first frame and the other in the second frame then an incorrect feature correlation would occur. On average this occurred with between 5-15% of the features on the various scenes and lighting conditions tested. Although the correlation scheme using the prioritized features gave good results, it was replaced by the template matching core because the template matching core proved not only to be more robust and but also able to correlate more features per frame than the priority queue correlation.

3.5.3 Template Matching

Figure 3.4 shows a feature pattern used in controlled lab experiments and the resulting correlated features. The green squares represent the Harris features detected in the current image and red squares appear where a match of that feature was found in the subsequent image. Experimental results showed that the template matching core was able to match a total of 2636 features per frame at 30 fps.

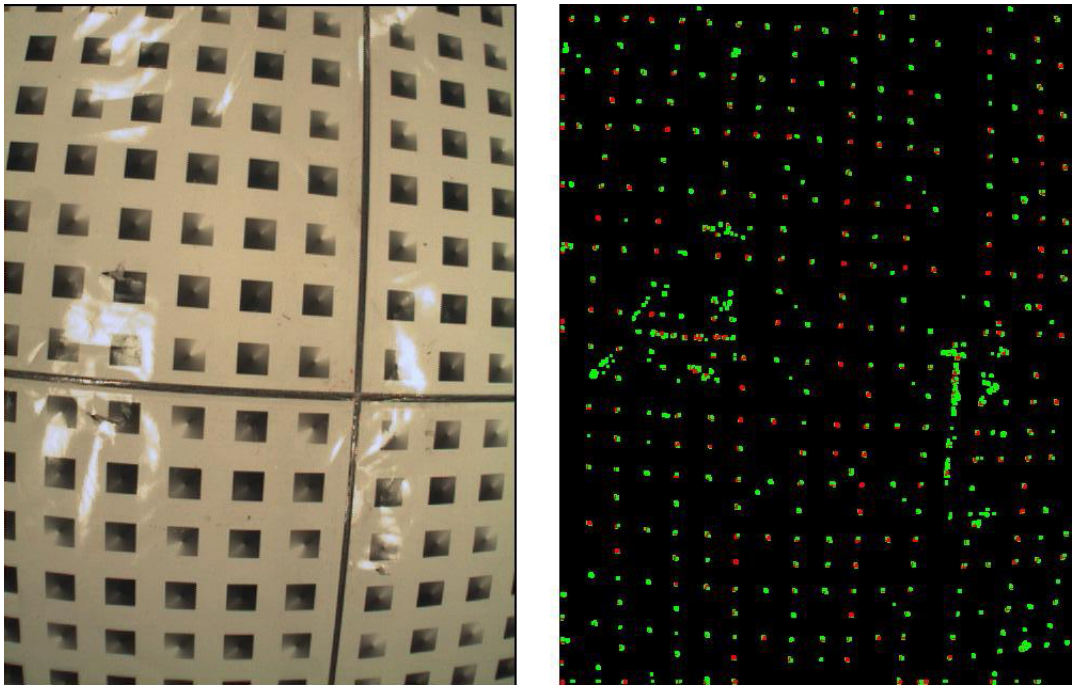


Figure 3.4: Template-matched feature scene.

Chapter 4

RANSAC Similarity-constrained Homography Estimation

4.1 Introduction

Once features are identified by the Harris feature detector core and correlated across images by the template matching core, a RANSAC similarity-constrained homography algorithm described in [37] is used to determine the translation, rotation, and scaling difference between the images. Details of how the RANSAC similarity-constrained homography algorithm was implemented on the Helios platform are given (Section 4.3), followed by experimental results (Section 4.4).

4.2 Background

There is extensive research showing the performance of RANSAC algorithms. Hartley and Zisserman showed that performing the RANSAC algorithm for the appropriate number of samples could guarantee a 99% probability of removing outliers [43]. Perez and Garcia show excellent results from a feature-based algorithm using RANSAC to estimate an similarity-constrained homography in their efforts to mosaic images [44]. Mallick shows RANSAC similarity-constrained homography estimation outperforming a least squares algorithm [45]. Thunuguntla and Gunturk show that a feature-based implementation of the RANSAC algorithm is robust for finding accurate homographies under rotation and scaling changes even in the presence of moving objects in the scene [46]. Work has also been done using the Harris Feature Detector and RANSAC algorithms together to generate accurate homographies [47]. Their results show that RANSAC algorithms are robust even when large disparities between images exist.

4.3 Algorithm Implementation

The similarity-constrained homography equations used in the RANSAC algorithm are derived from the similarity model shown in Equation (4.1), where s, θ, T_x, and T_y are the scale factor, angle of rotation, translation along the x axis, and translation along the y axis, respectively. Equation (4.2) shows the linear relation of Equations (4.3), (4.4) and (4.5), where (x_1, y_1) and (x_2, y_2) are two feature indices and $(x_1^{'}, y_1^{'})$ and $(x_2^{'}, y_2^{'})$ are the matching feature indices in a previous image. Elements C, D, T_x, and T_y of Equation (4.4) are defined by Equations (4.6), (4.7), (4.11), and (4.12).

$$\begin{bmatrix} x^{'} \\ y^{'} \end{bmatrix} = s \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}, \tag{4.1}$$

$$Ax = b, \tag{4.2}$$

$$A = \begin{bmatrix} x_1^{'} & -y_1^{'} & 1 & 0 \\ y_1^{'} & x_1^{'} & 0 & 1 \\ x_2^{'} & -y_2^{'} & 1 & 0 \\ y_2^{'} & x_2^{'} & 0 & 1 \end{bmatrix}, \tag{4.3}$$

$$x = \begin{bmatrix} C & D & T_x & T_y \end{bmatrix}^T, \tag{4.4}$$

$$b = \begin{bmatrix} x_1 & y_1 & x_2 & y_2 \end{bmatrix}^T. \tag{4.5}$$

Solving Equation (4.8) produces the determinant of matrix A ($det(A)$). Equations (4.9) and (4.10) can then be used to find the angle of rotation (θ), and the scale factor (s) between the two images.

$$C = \frac{x_1(x_1^{'} - x_2^{'}) + y_1(y_1^{'} - y_2^{'}) + x_2(x_2^{'} - y_1^{'}) + y_2(y_2^{'} - y_1^{'})}{det(A)}, \tag{4.6}$$

$$D = \frac{x_1(y_2^{'} - y_1^{'}) + y_1(x_1^{'} - x_2^{'}) + x_2(y_1^{'} - y_2^{'}) + y_2(x_2^{'} - x_1^{'})}{det(A)}, \tag{4.7}$$

$$det(A) = x_1^{'2} + y_1^{'2} + x_2^{'2} + y_2^{'2} - 2y_1^{'}y_2^{'} - 2x_1^{'}x_2^{'}, \tag{4.8}$$

$$\theta = -\arctan\left(\frac{D}{C}\right), \tag{4.9}$$

$$s = \frac{\cos(\theta)}{C}. \tag{4.10}$$

Translation along the two perpendicular axes, T_x and T_y, is calculated using Equations (4.11) and (4.12), which are derived by solving for T_x and T_y in Equation (4.4). Variables E, F, G, and H are defined in Equations (4.13), (4.14), (4.15), (4.16), respectively.

$$T_x = \frac{Ex_1 + Fy_1 + Gx_2 + Hy_2}{det(A)}, \tag{4.11}$$

$$T_y = \frac{-Fx_1 + Ey_1 + Hx_2 + Gy_2}{det(A)}, \tag{4.12}$$

$$E = x_2^{'2} + y_2^{'2} - y_1^{'}y_2^{'} - x_1^{'}x_2^{'}, \tag{4.13}$$

$$F = y_2^{'}x_1^{'} - y_1^{'}x_2^{'}, \tag{4.14}$$

$$G = x_1^{'2} + y_1^{'2} - y_1^{'}y_2^{'} - x_1^{'}x_2^{'}, \tag{4.15}$$

$$H = y_1^{'}x_2^{'} - y_2^{'}x_1^{'}. \tag{4.16}$$

Just like a traditional RANSAC algorithm, random sets of feature indices are selected and the similarity-constrained homography results that are calculated from them are compared with the first set using a Euclidean distance measure. Of the set of inliers that are produced from the RANSAC voting, the one pair of points that has the most votes is selected as the pair that most closely represents the whole set of inliers. The similarity-constrained homography generated by this pair is used for the helicopter command. Using one pair of points reduces the amount of computation needed to generate a similarity-constrained homography by reducing an over-constrained problem of multiple points to a problem with an exact solution.

4.4 Results

4.4.1 RANSAC Homography

Three sequences of images and resulting feature points were captured of controlled scene movements and evaluated in MATLAB. The similarity-constrained homography calculations for each pair of images were accumulated in order to compare estimated scene movement over the sequence with the actual movement. Tests involving just rotation angle, then scale factor, and finally translation along just the x axis were performed. When plotted, the RANSAC similarity-constrained homography algorithm results represent the actual movement of the scene in each the three tests well (see Figure 4.1). For the rotation test, the scene was rotated 636 degrees in a series of revolutions. The RANSAC algorithm accumulated a measurement of 596 degrees, resulting in an accumulated error of 6.29%. The scene was moved toward the camera, away from it, and then back towards it for the test involving just changes in scale for a total accumulated change in scale factor of 4.08. The algorithm estimated a total of 4.20, resulting in an error of 2.69%. The translation-only test involved moving the scene back and forth two times for a total distance of 225 pixels, which resulted in an error of 2% with the RANSAC estimation of 220 pixels. flight test results showed that the error evident in these tests did not negatively affect stable control of the Helio-copter.

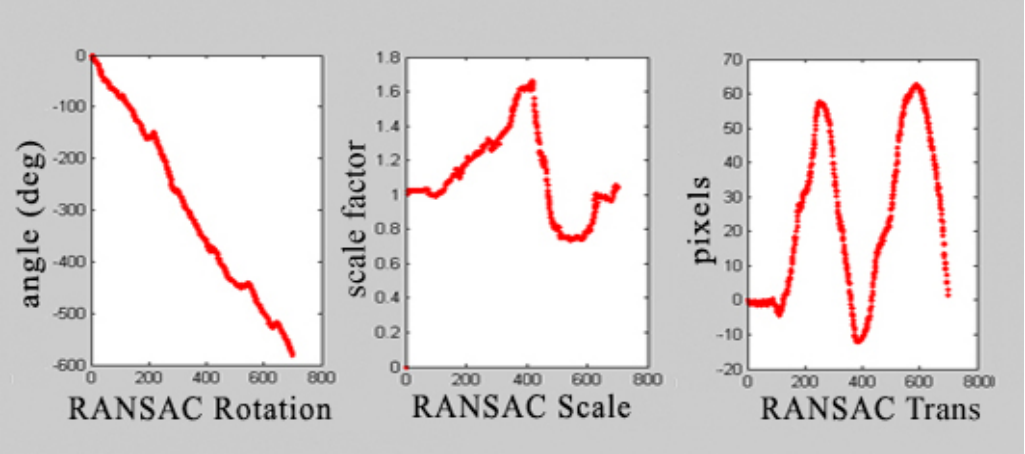


Figure 4.1: RANSAC similarity-constrained homography test results. Graphs are results of rotation, scale, and x translation tests.

Throughout testing it was observed that inaccuracies in the homography were evident when too few features existed in the scene. Also, the accuracy of a homography is obviously dependent on the resolution of the camera used since the homography is calculated using pixel indices.

4.4.2 Helio-copter Drift Stabilization

Implementing these vision algorithms allowed the Helio-copter to successfully detect when it was drifting. The Helio-copter control gains were adjusted so that the Helio-copter maintained very steady positions for yaw and altitude. Corrections for translational drift required the Helio-copter to pitch and roll causing the camera to see even more translation of the scene and amplifying the correction command. This made tuning the control gains for pitch and roll commands much more difficult, and to minimize the effect the gains were kept low allowing the Helio-copter to move more over a scene. The Helio-copter was still able to hold a fixed position within a six foot by six foot area for up to 43 seconds before it eventually worked itself outside of this area.

Chapter 5

Target Tracking

5.1 Introduction

After achieving stable hovering flight over a feature scene, the project focus was redirected to maintaining stable attitude while tracking a target. In maintaining a stable position over a fixed scene, features throughout the whole image were used to correctly estimate homographies between images. To follow a moving target requires the image processing hardware to ignore the translation of all features in the image except those defined as being the desired target. For practical applications a quad-rotor would need to be able to hold a position over a scene until a user selected a target in the image scene at which point the vision system would switch tasks and use features of the target to hold its position, moving if the target moved. As a proof-of-concept of the ability of the vision and control systems of the Helio-copter to perform a target tracking task, a target of two colored dots was used. This controlled test target allowed a display of the capabilities of the systems by implementing color space conversion (Section 5.3.1), a color segmentation algorithm (Section 5.3.2), connected components algorithm (Section 5.3.4), radial distortion correction (Section 5.4), and a Kalman filter (Section 5.6) using a standard-deviation outlier rejection technique (SORT) (Section 5.5). A block diagram of this system is shown in Figure 5.1.

5.2 Background

5.2.1 Color Space Conversion and Segmentation

Segmenting colors using the HSV color space allows for a more robust solution under varying lighting conditions [48]. This implementation uses the RGB to HSV

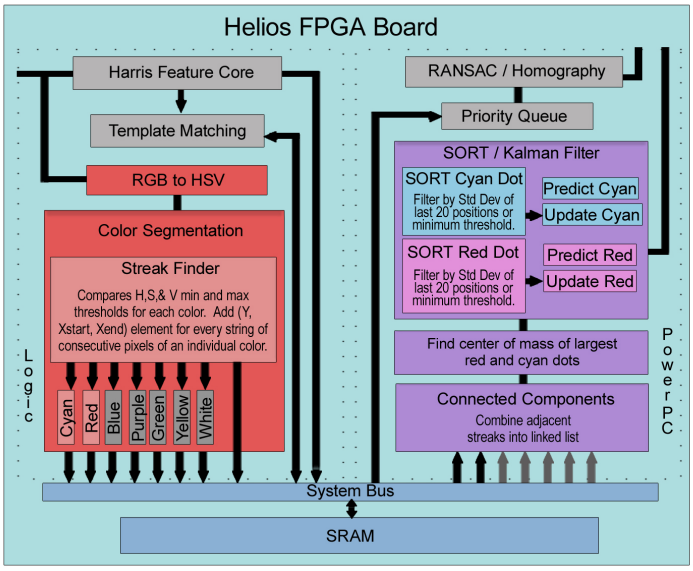


Figure 5.1: Data flow through components of the target tracking system.

conversion equations described by Travis [49]. Newton Labs offers hardware implementing a real-time constant threshold segmentation algorithm for the RGB color space[50]. Often specialized hardware is required to achieve real-time performance for multiple color segmentation, although optimized algorithms have been designed to achieve similar performance in software [51].

5.2.2 Radial Distortion

In the presence of radial and tangential distortion when using a wide angle lens, calibration techniques can be used to calculate the coefficients of the forward mapping equations going from the undistorted image to the distorted image. Heikkila and Silven point out that there is no general algebraic expression for the inverse mapping from the distorted image to the undistorted image [52]. A Matlab toolkit for finding camera calibration parameters is available from Caltech [53].

5.2.3 Kalman Filter

The Kalman filter supports estimations of present and future states even when the precise nature of the modeled system is unknown [54]. Earl and D'Andrea achieved

successful attitude estimation results for a quad-rotor by decomposing on-board rate gyro measurements and off-board vision sensor measurements and then applying a Kalman filter [17].

5.2.4 SORT

Zhang et al. describe the range method, a common univariate statistics analysis technique, where the mean and standard deviation of the dataset are used to label each element in the dataset an inlier or an outlier. Values of the dataset that are outside the limits of the mean plus or minus n times the standard deviation ($mean \pm n \times stdev$) are considered to be the outliers [55].

5.3 Segmenting Connected Components

The color segmentation and connected component analysis system is divided into hardware and software elements. The color segmentation FPGA core includes an RGB to HSV color space conversion block, a hue, saturation, and intensity threshold block, and a block that identifies streaks of contiguous pixels having the same color. Structures containing "streak" information are stored in memory and then linked together in a list by software. The center of each of the colored connected components is also calculated in software.

5.3.1 Color Space Conversion

The HSV color space conversion was pipelined into five stages in the FPGA hardware. Implementing this color space conversion in hardware adds five clock cycles of latency. The hardware implementation is substantially faster than any possible software version, as processing a single image would take about eleven million sequential instructions in software. The HSV values for each pixel are calculated from the RGB values normalized to the range (0,1) using Equations (5.1), (5.2), and (5.3), where MAX is assigned either R, G, or B, which ever has the largest value, and MIN is assigned whichever has the lowest value of R, G, and B.

$$H = \begin{cases} \text{undefined} & \text{if MAX} = \text{MIN} \\ 60^\circ \times \frac{G-B}{MAX-MIN} & \text{if MAX} = \text{R and G} \geq \text{B} \\ 60^\circ \times \frac{G-B}{MAX-MIN} + 360^\circ & \text{if MAX} = \text{R and G} < \text{B} \\ 60^\circ \times \frac{B-R}{MAX-MIN} + 120^\circ & \text{if MAX} = \text{G} \\ 60^\circ \times \frac{R-G}{MAX-MIN} + 240^\circ & \text{if MAX} = \text{B} \end{cases}, \tag{5.1}$$

$$S = \begin{cases} 0 & \text{if MAX} = 0 \\ 1 - \frac{MIN}{MAX} & \text{otherwise} \end{cases}, \tag{5.2}$$

$$V = MAX. \tag{5.3}$$

5.3.2 Color Segmentation

Once each pixel has been converted to the HSV color space, it must be compared against upper and lower thresholds for hue, saturation, and intensity. The threshold values are set manually allowing for differences in camera color calibrations. If the incoming pixel is within the thresholds, then the color segmentation block labels that pixel as a 1. If the pixel values are not within the thresholds, the pixel is labeled with a 0. This stream of pixel data is then passed on to the streak finder core and into FIFOs which can be read by a PLB read burst and written into memory.

Performing this operation in hardware takes advantage of the ability to segment as many colors as needed simultaneously, all while only adding a few more stages to the image pipeline. This reduces a potential delay of as much as half a second for a software implementation to less than 20 clock cycles of added latency.

5.3.3 Streak Finder

The binary output from the segmentation block is used in the streak finder block to identify continuous segments of the same color in a row of the image. When the first binary value of one is received, both the y index and x index are stored in registers. These are saved if subsequent pixel values of one are read. As soon as a

zero is received or the end of the row is reached, then the core combines the registered y and x indices and the current x index and puts them in a fifo that will be burst into memory and made available to the software process. These values are eventually read into a streak structure containing a row index, the starting x index, and an ending x index.

Breaking up a connected components algorithm in this manner and implementing this streak finder in hardware reduces hundreds of milliseconds of processing time in software to just a few extra clock cycles of latency. When the streak structures are read from memory, they can easily be linked together in a linked list in software.

5.3.4 Connected Components

The connected components algorithm is completed in software by grouping together the streaks that were found in hardware. A streak is added to an existing linked list if it lies in an adjacent row and has overlapping x starting and ending indices of another streak in the list. If no list with an adjacent streak is found then a new list is created. This type of repeated list comparing algorithm is better suited for implementation in software than hardware.

As linked lists of streaks are created, minimum and maximum x and y indices are kept so that a center of mass (x_c, y_c) can be computed. Since the connected components used in this project are circles, the center of mass can be found using the equations in (5.4). The number of pixels in each streak in a connected component is also accumulated and stored as its mass, which can be thresholded for noise rejection.

$$\begin{aligned} x_c &= (x_{min} + x_{max})/2, \\ y_c &= (y_{min} + y_{max})/2. \end{aligned} \tag{5.4}$$

For this project two colors were segmented: red and cyan. The center of mass of each connected component was used for the similarity-constrained homography calculation. Using two points is a minimum for a similarity-constrained homography calculation and it was observed that certain cases would cause the system to not

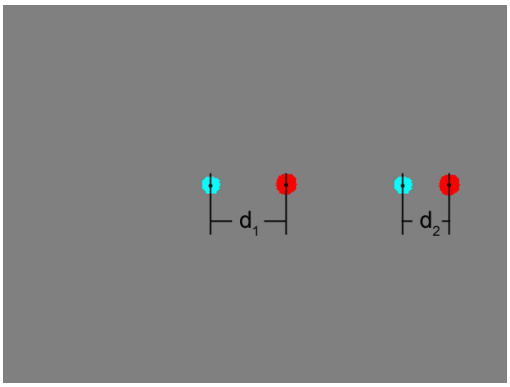


Figure 5.2: Radial distortion correction. Distance between dots shrinks as they move away from the center of the image when the altitude of the Helio-copter is constant.

detect one of the connected components or misrepresent the locations of the centers of mass. To correct for these cases, three algorithms were implemented and added to the system before the similarity-constrained homography was calculated. These algorithms included a radial distortion correction, an outlier rejector, and a Kalman filter.

5.4 Correcting Radial Distortion

The lens used on the image sensor for the target tracking task is a wide angle 2.1mm lens which allows an area of about six feet by six feet to be visible from an altitude of approximately six feet. This lens has a wider field of view but also warps the image towards the edges. The distortion in the image affects the scale factor of the similarity-constrained homography. Basing the similarity-constrained homography on a minimum number of two points causes the scale factor to be simply the distance between the two points divided by an absolute reference distance. As the Helio-copter translates across the target, the radial distortion of the camera lens causes the segmented dot to appear closer to the center of the image than it really is. This affects the scale factor when one dot is found farther from the center of the image than the other, causing it to be shifted by distortion more, as seen in Figure 5.2.

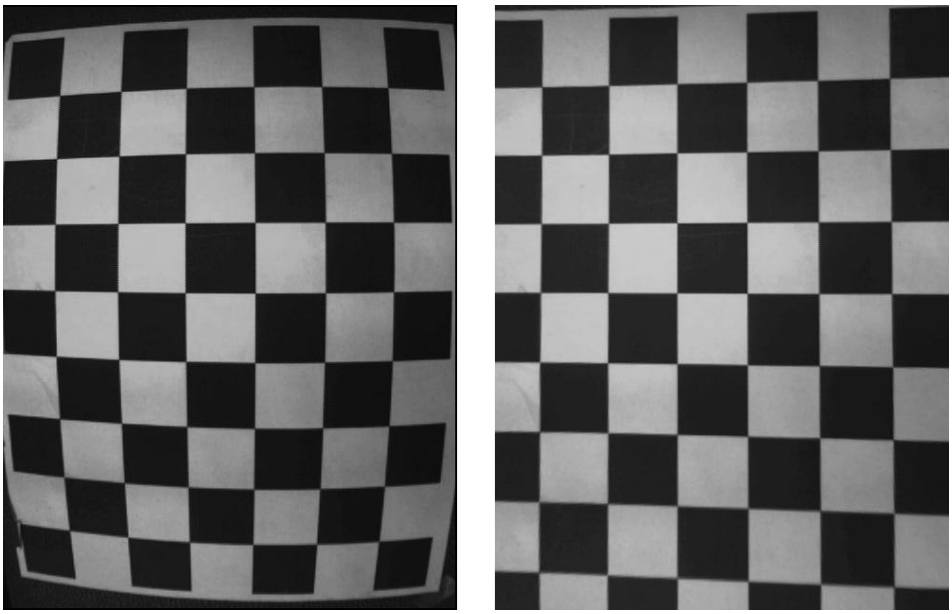


Figure 5.3: Radially distorted image compared to the undistorted image.

The Caltech Camera Calibration toolbox for MATLAB was used to generate an undistorted image from a distorted image taken with the Helio-copter image sensor. The original image and undistorted image are shown in Figure 5.3. The pixel indices of the grid intersections in both images were determined and correlated manually by observation. The resulting set of paired points were used to generate a best fit polynomial equation with the polyfit function in MATLAB. This polynomial provided an estimated mapping of points from the distorted image to the undistorted image. This approximation was used because of the lack of a general algebraic expression for the inverse mapping of distorted image points to their undistorted locations.

After the coefficients α_1, α_2, and α_3 of the quadratic equation were found, then the undistorted point (x', y') could be found from the distorted point (x, y) using Equations (5.6) through (5.9). Radial distortion occurs with respect to the center of the image, and the pixel indices are registered in the hardware core with respect to the upper left hand corner of the image. The origin must be shifted before the points are undistorted, Equation (5.5), and then returned to the original origin afterward, Equation (5.10), where $WIDTH$ and $HEIGHT$ are the width and height of the image.

$$\begin{aligned} x &= x - \tfrac{WIDTH}{2}, \\ y &= \tfrac{HEIGHT}{2} - y, \end{aligned} \tag{5.5}$$

$$r' = \alpha_1(x^2 + y^2) + \alpha_2\sqrt{x^2 + y^2} + \alpha_3, \tag{5.6}$$

$$\theta = \arctan\frac{y}{x}, \tag{5.7}$$

$$y' = \begin{cases} r'\sin\theta & \text{if x} \geq 0 \\ -r'\sin\theta & \text{if x} < 0 \end{cases}, \tag{5.8}$$

$$x' = \begin{cases} r'\cos\theta & \text{if x} \geq 0 \\ -r'\cos\theta & \text{if x} < 0 \end{cases}, \tag{5.9}$$

$$\begin{aligned} x &= \tfrac{WIDTH}{2} + x, \\ y &= \tfrac{HEIGHT}{2} - y. \end{aligned} \tag{5.10}$$

5.5 Standard-deviation Outlier Rejection Technique

After correcting for radial distortion, calculating a similarity-constrained homography using two points proved sufficient except when one of the points was not detected correctly. This occurred sometimes when the Helio-copter or the target moved quickly enough that a dot left the image before the Helio-copter was able to recover, or an object temporarily impeded the view of a dot. To increase the robustness of the target tracking system, a Kalman filter was implemented. Given the fact that a Kalman filter does not in and of itself detect outliers, a standard-deviation outlier rejection technique (SORT) was used in conjunction with it.

The SORT algorithm stores the previous 20 locations of both the red and cyan dots. The standard deviation of each set of 20 points is calculated and used to threshold the newest dot locations. As mentioned, the range method uses the $mean \pm n \times stdev$ as the threshold to label outliers. A value of $n = 3$ was found empirically to work well. Through experimental results it was determined that the

magnitude of $n \times stdev$ also needed to be saturated to a fixed lower limit to prevent it from getting too small if the platform didn't move much.

5.6 Kalman Filter

A Kalman filter is able to use a basic physics model to estimate the center of mass of a dot from its previous location when it is not detected correctly. The Kalman filter ends up smoothing out changes in the locations of the dots, and prevents drastic changes that could cause the Helio-copter to become unstable, which still allows it to follow the target without too much lag.

In this implementation, when a point has been classified as an outlier, the Kalman filter responds by modifying the sensor update covariance matrix so as to be less confident in that point. An Extended Kalman filter was implemented independently for each dot, with separate prediction and updating stages for the red and cyan dots, as shown in Figure 5.1. Each filter's state vector (X_k) was composed of the x and y positions of the dot it was filtering (Equation (5.11)). The sensor input (Z_k) to the filter is also the x and y positions of each dot, making the H matrix of the Kalman filter update equations in (5.12) the identity matrix.

$$X_k^{color} = \begin{bmatrix} x^{color} \\ y^{color} \end{bmatrix} \tag{5.11}$$

$$\begin{aligned} \hat{Y}_k &= Z_k - H_k \hat{X}_{k|k+1}, \\ S_k &= H_k P_{k|k-1} H_k^T + R_k, \\ K_k &= P_{k|k-1} H_k^T S_k^{-1}, \\ \hat{X}_{k|k} &= \hat{X}_{k|k-1} + K_k \hat{Y}_k, \\ P_{k|k} &= (I - K_k H_k) P_{k|k-1}. \end{aligned} \tag{5.12}$$

The equations are then simplified even further by breaking all matrix operations into single element operations shown in Equations (5.13).

$$
\begin{aligned}
\hat{y}(1)_k &= z(1)_k - \hat{x}(1)_{k|k+1}, \\
\hat{y}(2)_k &= z(2)_k - \hat{x}(2)_{k|k+1}, \\
s(1)_k &= p(1,1)_{k|k-1} + r(1)_k, \\
s(2)_k &= p(2,2)_{k|k-1} + r(2)_k, \\
k(1)_k &= \frac{p(1,1)_{k|k-1}}{s(1)_k}, \\
k(2)_k &= \frac{p(2,2)_{k|k-1}}{s(2)_k}, \\
\hat{x}(1)_{k|k} &= \hat{x}(1)_{k|k-1} + k(1)_k \hat{y}(1)_k, \\
\hat{x}(2)_{k|k} &= \hat{x}(2)_{k|k-1} + k(2)_k \hat{y}(2)_k, \\
p(1,1)_{k|k} &= p(1,1)_{k|k-1} - k(1)_k p(1,1)_{k|k-1}, \\
p(2,2)_{k|k} &= p(2,2)_{k|k-1} - k(2)_k p(2,2)_{k|k-1}.
\end{aligned}
\tag{5.13}
$$

The predict stage function for the cyan dot is shown in Equation (5.15). This part of the Kalman filter is what allows the Helio-copter to track the target for short times when one of the dots is not detected. Normally the filter predicts the position of the dot using the velocity of the dot calculated from its current position and its position four frames prior (Equation (5.14)), but when a dot is not detected, then it will use the velocity of the other dot. The success of this algorithm is based on the assumption that if the Helio-copter loses a dot, it will not drastically change altitude or heading before the dot is recovered. Experimental results are discussed to support the validity of this assumption (Section 5.7.3).

$$
\begin{aligned}
v_k^x &= \frac{x_k - x_{(k-4)}}{4}, \\
v_k^y &= \frac{y_k - y_{(k-4)}}{4}.
\end{aligned}
\tag{5.14}
$$

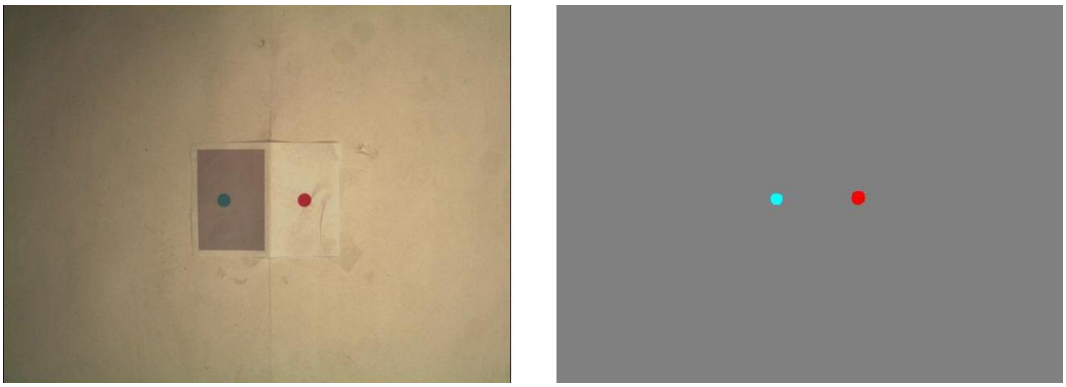


Figure 5.4: A sample scene and the resulting segmented target dots.

$$\begin{bmatrix} x_k^{cyan} \\ y_k^{cyan} \end{bmatrix} = \begin{cases} \begin{bmatrix} x_{k-1}^{cyan} + v_{k-1}^{xcyan} \\ y_{k-1}^{cyan} + v_{k-1}^{ycyan} \end{bmatrix}, & \text{if } dot^{cyan} \neq \text{outlier} \\ \\ \begin{bmatrix} x_{k-1}^{cyan} + v_{k-1}^{xred} \\ y_{k-1}^{cyan} + v_{k-1}^{yred} \end{bmatrix}, & \text{otherwise} \end{cases} . \tag{5.15}$$

5.7 Results

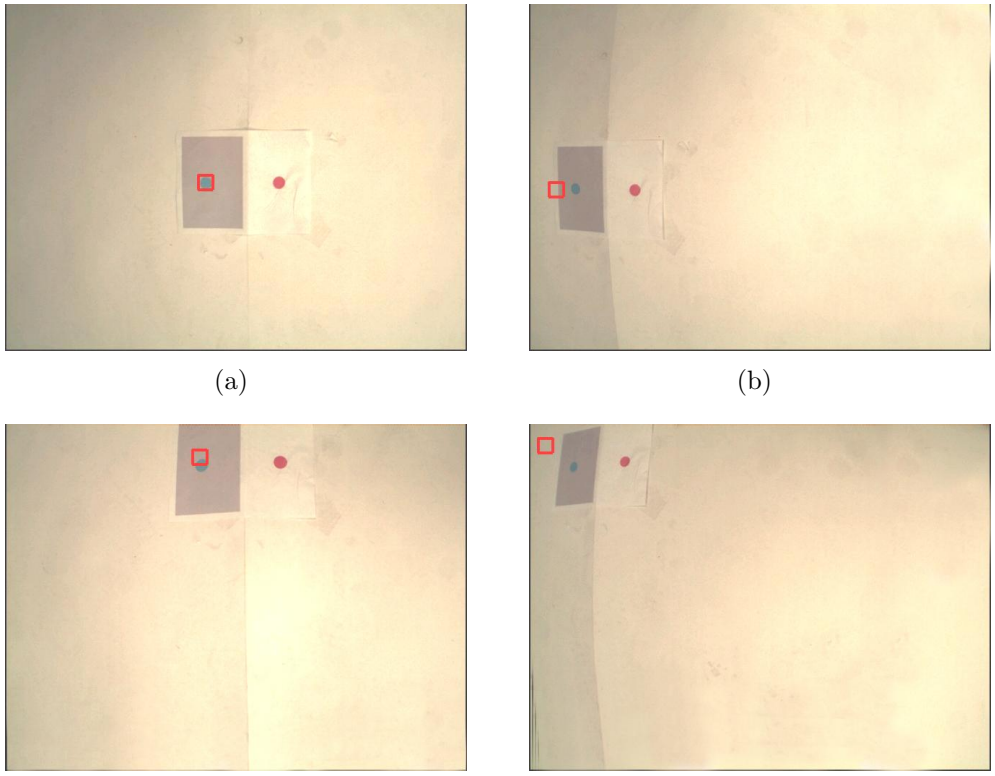
5.7.1 Color Space Conversion and Segmentation

Implementing the color space conversion and segmentation algorithms in hardware, where they could be placed in the pixel data pipeline, turned a potentially computation-time intensive process into an added latency of less than a dozen clock cycles. The ability to manually adjust thresholds makes it dynamic and suitable for many applications. Figure 5.4 shows a sample image of the target dots and the binary image of the two segmented colors cyan and red.

5.7.2 Radial Distortion Correction

After the approximation equation for inverse mapping of distorted image points to undistorted image points was implemented, flight tests were performed to ensure proper performance. Four sample images with a box denoting the undistorted location of the cyan dot were captured on the Helio-copter (Figure 5.5). Flight tests

involving translation of the Helio-copter over the two-dot target showed that altitude changes were negligible anywhere in the field of view of the lens.



(a) (b)

(c) (d)

Figure 5.5: Four dots with radial distortion correction. Four sample scenes overlaid with the undistorted location of the red dot in four different locations in the image. (a) Almost no correction near the image center. (b) Larger corrections towards edge of image. (c) Distortion mostly along the y axis. (d) Distortion along both x and y axes.

5.7.3 Kalman Filter and SORT

The extended Kalman filter was simulated in MATLAB using actual positions and velocities of the target dots logged during a Helio-copter test flight. After achieving successful simulation results, the filter was implemented on the Helios board. Once again, dot positions were logged, this time with the Kalman filtered points as well. The logged values were taken from a test where one of the dots leaves the image

for a period of time and re-enters at a different point on the image. The logged data for one of the two dots was then plotted in MATLAB with red triangles denoting measured dot positions from the image sensor, and green crosses denoting Kalman filter estimations. Figure 5.6 shows this plot. As can be seen, the Kalman filter followed the measured locations of the dot through a full circle, starting at the top and just to the left of the middle of the circle. As the dot moved out of the field of view of the camera on the left side the measured position of the dot was reported as zero, denoted by the missing red triangles along the left side of the figure. From this point on the Kalman filter continued to estimate the location of the dot using the velocity vector of the visible dot. When the dot re-entered the view of the image sensor (lower right hand corner of the figure), the Kalman filter estimations were close enough to the actual location of the dot that the filter could recover without making large jumps in the estimated dot location. This performance proved the robustness of this Kalman filter implementation to losing dots from the image, providing a greater range of stability to the Helio-copter than it had with just the field of view of the camera.

Various tests were performed to evaluate the effectiveness of SORT in detecting outliers. A sample case where the target moved very abruptly caused the algorithm to reject the large change at first, but when enough measurements verified the new location, the standard deviation increased until the new location was not considered an outlier and the filter-estimated dot positions followed a smooth yet quick change to update to the new location. The filter also showed a resistance to large accelerations and random jumps as expected. The implementation of the extended Kalman filter with SORT significantly increased the robustness of the target tracking system on the Helio-copter.

5.7.4 Helio-copter Performance

Flight tests were performed where the Helio-copter was to hold a fixed position over the stationary two-dot target. Although the Helio-copter was able to hold position really well for yaw and altitude, similar behavior was observed during tests

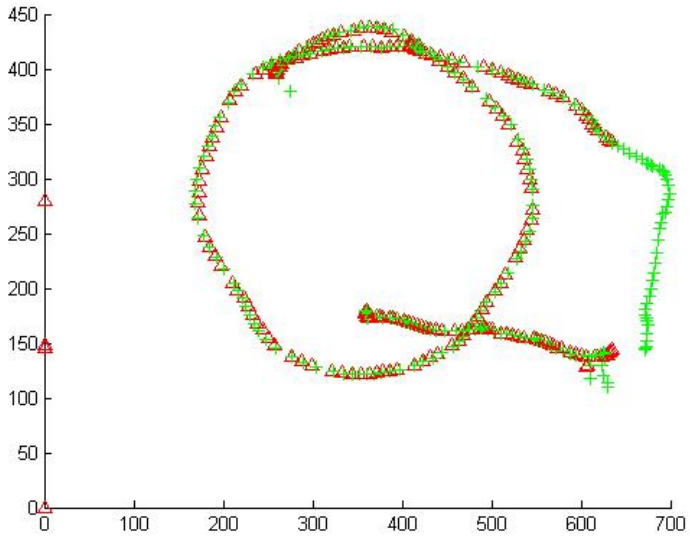


Figure 5.6: Kalman filter results. Kalman filter performance in the case where a dot is lost from the image (red triangles are missing). The position of the occluded dot is calculated using its previous position and the velocity of the visible dot, preserving the distance relationship between the two dots.

over the target as was observed with tests over the feature scene when trying to prevent translational drift. In the case of tracking a target, the small field of view of the camera at the current testing height combined with the low control gains on pitch and roll commands prevented the Helio-copter from correcting enough to not lose the target. Other approaches such as testing with a wider angle lens, or combining vision information and IMU information to prevent overcorrection are being considered to obtain longer, more stable flight results.

Chapter 6

Conclusion

6.1 Platform Resource Allocation

Successful implementation of an on-board vision system on the Helio-copter left substantial resources available for future work. As can be seen in Table 6.1, only about 1/3 of the Virtex-4 FPGA is utilized when considering that the design was not optimized for area, allowing routing to use far more slices than are needed. The true exceptions to this are the number of I/O buffers and BRAMs. Even more programmable logic is available if unused resources on the Spartan FPGA on the AVT daughterboard are considered. Figure 6.1 also indicates the possible capabilities of the Helio-copter vision system by comparing resource usage of individual algorithms. Table 6.2 shows the percentage of available code space used in the two processors, Helios' PowerPC and KAP's Rabbit, to implement overall control and the software component of each algorithm. The low use of computational resources for this project indicate that the Helio-copter vision system has great potential for implementing additional vision alogrithms.

Table 6.1: Helios FPGA resource usage.

	Helios Virtex FX60		
	Used	Capacity	%
Slice Flip Flops	16,784	50,560	33%
4 input LUTs	19,236	50,560	38%
Slices	15,818	25,280	62%
Bonded IOBs	200	352	56%
RAMB16 / BRAM	129	232	55%
DSP48s	39	128	30%

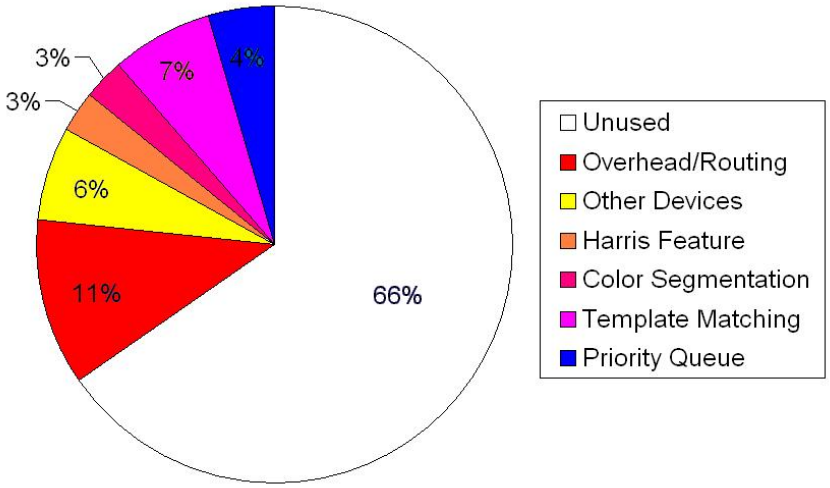


Figure 6.1: Usage breakdown of the Helios FPGA resources with algorithms implemented for this project.

Table 6.2: Code distribution.

	Used(KB)	Capacity(KB)
KAP	127.74	256
Helios PowerPC	65.5	128

6.2 Quad-rotor Platform

The Helio-copter meets the design specifications that were detailed at the outset of the design process. It is capable of carrying all necessary vision and control electronics, two batteries that last for close to 20 minutes of flight, as well as motor capabilities of almost an extra two pounds of payload. If extra payload is used for more batteries, the Helio-copter should be able to obtain flight times in excess of 30 minutes, even at the current elevation of 4500 ft.

6.3 Real-Time On-board Vision Algorithms

Hardware/Software co-designs of a Harris feature detector, a template matching feature correlator, a RANSAC homography algorithm, color space conversion, a color segmentation algorithm, a connected components algorithm, radial distortion

correction, and a Kalman filter using a standard-deviation outlier rejection technique(SORT) were implemented achieving real-time performance. These algorithms were incorporated into a quad-rotor platform to overcome attitude estimation issues and obtain stable hovering capabilities.

6.4 Project Contributions

This project makes the following contributions:

1. **Quad-rotor Platform**

   Improved quad-rotor platform - A lightweight, robust, powerful quad-rotor platform design is tested and proven.

2. **Hardware vision algorithms**

   Harris feature detector FPGA core - A template matching hardware core coupled with a Harris feature core was also tested and incorporated into the Helio-copter vision system.

   Feature priority queue FPGA core - This core was implemented and tested and will be considered for incorporation into the Helio-copter vision system in the future.

3. **Software Algorithms**

   RANSAC homography algorithm - This algorithm was implemented and achieved real-time performance on the Helios' PowerPC running at 100 MHz.

   Extended Kalman filter with SORT - Implementation increased robustness of target tracking.

4. **Experimental Results**

   Helio-copter - Stable flight of the Helio-copter was achieved for short periods of time through numerous tests to obtain proper control parameters.

6.5 Future Work

Quad-rotor platforms provide interesting control challenges and impressive application possibilities. Continuing work started in this project will include implementing a wireless analog video transmitter to have live video feedback to the ground station. This would allow for mid-flight mission modifications as well as mid-flight debugging capabilities by transmitting images from any intermediate step of processing.

Development of support for two cameras will continue so that stereo vision algorithms can be implemented on the platform. Dual camera support will also provide the ability to process images of different views, i.e., one downward-facing camera aids in attitude estimation, and a second forward-facing camera performs obstacle detection. Although there has not been any fully autonomous quad-rotor flight documented to date, the author feels that quad-rotor research is close to producing a fully autonomous hovering platform ready for application in many of the suggested fields.

Bibliography

[1] F. Archer, A. Shutko, T. Coleman, A. Haldin, E. Novichikhin, and I. Sidorov, "Introduction, overview, and status of the microwave autonomous copter system (MACS)," *Geoscience and Remote Sensing Symposium, 2004. IGARSS '04. Proceedings. 2004 IEEE International*, vol. 5, pp. 3574–3576 vol.5, 20-24 Sept. 2004. 1

[2] R. Sugiura, T. Fukagawa, N. Noguchi, K. Ishii, Y. Shibata, and K. Toriyama, "Field information system using an agricultural helicopter towards precision farming," in *Advanced Intelligent Mechatronics, 2003. AIM 2003. Proceedings. 2003 IEEE/ASME International Conference on*, vol. 2, 20-24 July 2003, pp. 1073–1078. 1

[3] S. Rock, E. Frew, H. Jones, E. LeMaster, and B. Woodley, "Combined CDGPS and vision-based control of a small autonomous helicopter," in *American Control Conference, 1998. Proceedings of the 1998*, vol. 2, 24-26 June 1998, pp. 694–698. 1

[4] E. Altüg, J. Ostrowski, and R. Mahony, "Control of a quadrotor helicopter using visual feedback," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, vol. 1, 11-15 May 2002, pp. 72–77. 1, 7

[5] L. Jun, X. Shaorong, G. Zhenbang, and R. Jinjun, "Subminiature unmanned surveillance aircraft and its ground control station for security," in *Safety, Security and Rescue Robotics, Workshop, 2005 IEEE International*, 6-9 June 2005, pp. 116–119. 1

[6] C. A. Patel, "Building a testbed for mini quadrotor unmanned aerial vehicle with protective shroud," Master's thesis, Wichita State University, Dept. of Mechanical Engineering, May 2006. 1, 8

[7] M. Koeda, Y. Matsumoto, and T. Ogasawara, "Development of an immersive teleoperating system for unmanned helicopter," in *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*, 25-27 Sept. 2002, pp. 47–52. 2

[8] P. Pounds, R. Mahony, and P. Corke, "Modelling and control of a quad-rotor robot," in *Australasian Conference on Robotics and Automation 2006*, December 2006. 4, 12

[9] S. Bouabdallah, M. Becker, and R. Siegwart, "Autonomous miniature flying robots: Coming soon!" *IEEE Robotics & Automation Magazine*, pp. 88–98, September 2007. 4, 8, 12

[10] M. A. bin Ramli, C. K. Wei, and G. Leng, "Design and development of an indoor UAV," in *RSAF Aerospace Technology Seminar*, 2007. 4, 12

[11] S. Ettinger, M. Nechyba, P. Ifju, and M. Waszak, "Vision-guided flight stability and control for micro air vehicles," in *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, vol. 3, 30 Sept.-5 Oct. 2002, pp. 2134–2140. 5, 30

[12] G. Hoffmann, D. G. Rajnarayan, S. L. Waslander, D. Dostal, J. S. Jang, and C. J. Tomlin, "The Stanford testbed of autonomous rotorcraft for multi-agent control (STARMAC)," in *Proceedings of the 23rd Digital Avionics Systems Conference*, Salt Lake City, UT, November 2004, pp. 12.E.4/1–10. 5, 12, 15

[13] A. Neff, D. Lee, V. Chitrakaran, D. Dawson, and T. Burg, "Velocity control for a quad-rotor UAV fly-by-camera interface," in *SoutheastCon, 2007. Proceedings. IEEE*, 22-25 March 2007, pp. 273–278. 5, 7, 30

[14] V. Chitrakaran, D. Dawson, J. Chen, and M. Feemster, "Vision assisted autonomous landing of an unmanned aerial vehicle." in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, 12-15 Dec. 2005, pp. 1465–1470. 5, 7

[15] J.-C. Zufferey and D. Floreano, "Toward 30-gram autonomous indoor aircraft: Vision-based obstacle avoidance and altitude control," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, 18-22 April 2005, pp. 2594–2599. 5, 7

[16] E. Altüg, J. Ostrowski, and C. Taylor, "Quadrotor control using dual camera visual feedback," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 3, 14-19 Sept. 2003, pp. 4294–4299. 5, 7, 30

[17] M. Earl and R. D'Andrea, "Real-time attitude estimation techniques applied to a four rotor helicopter," in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 4, 14-17 Dec. 2004, pp. 3956–3961. 5, 7, 51

[18] J. Roberts, P. Corke, and G. Buskey, "Low-cost flight control system for a small autonomous helicopter," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 1, 14-19 Sept. 2003, pp. 546–551. 5

[19] S. Fowers, "Stabilization and control of a quad-rotor micro-UAV using vision sensors," Master's thesis, Brigham Young University, April 2008. 7, 25

[20] G. Barrows, "Future visual microsensors for mini/micro-UAV applications," in *Cellular Neural Networks and Their Applications, 2002. (CNNA 2002). Proceedings of the 2002 7th IEEE International Workshop on*, 22-24 July 2002, pp. 498–506. 7

[21] H. Romero, R. Benosman, and R. Lozano, "Stabilization and location of a four rotor helicopter applying vision," in *American Control Conference, 2006*, 14-16 June 2006, p. 6. 7, 30

[22] O. Shakernia, Y. Ma, T. Koo, and S. Sastry, "Landing an unmanned air vehicle: Vision based motion estimation and nonlinear control," 1999. [Online]. Available: citeseer.ist.psu.edu/shakernia99landing.html 8

[23] C. Sharp, O. Shakernia, and S. Sastry, "A vision system for landing an unmanned aerial vehicle," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 2, 2001, pp. 1720–1727. 8

[24] J. Escareno, S. Salazar-Cruz, and R. Lozano, "Embedded control of a four-rotor UAV," in *American Control Conference, 2006*, 14-16 June 2006, p. 6. 8

[25] S. L. Waslander, G. M. Hoffmann, J. S. Jang, and C. J. Tomlin, "Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmonton, AB, Canada, August 2005, pp. 468–473. 8

[26] P. Castillo, A. Dzul, and R. Lozano, "Real-time stabilization and tracking of a four-rotor mini rotorcraft," in *Control Systems Technology, IEEE Transactions on*, vol. 4, July 2004, pp. 510–516. 8, 12

[27] J. Dunfied, M. Tarbouchi, and G. Labonte, "Neural network based control of a four rotor helicopter," in *Industrial Technology, 2004. IEEE ICIT '04. 2004 IEEE International Conference on*, vol. 3, 8-10 Dec. 2004, pp. 1543–1548. 8

[28] W. S. Fife and J. K. Archibald, "Reconfigurable on-board vision processing for small autonomous vehicles," *EURASIP Journal on Embedded Systems*, vol. 2007, pp. Article ID 80 141, 14 pages, 2007, doi:10.1155/2007/80141. 9, 21

[29] R. Beard, D. Kingston, M. Quigley, D. Snyder, R. Christiansen, W. Johnson, T. McLain, and M. Goodrich, "Autonomous vehicle technologies for small fixed wing UAVs," *AIAA Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. 1, pp. 92–108, January 2005. 9, 24

[30] I. Kroo, F. Prinz, and M. Shantz, "The mesiocopter: A miniature rotorcraft concept," July 2000. 12

[31] G. P. Tournier, M. Valenti, J. P. How, and E. Feron, "Estimation and control of a quadrotor vehicle using monocular vision and Morié patterns," in *AIAA Guidance, Navigation, and Control Conference and Exhibit.* AIAA, August 2006. 12

[32] W. MacLean, "An evaluation of the suitability of FPGAs for embedded vision systems," in *Computer Vision and Pattern Recognition, 2005 IEEE Computer Society Conference on*, vol. 3, 20-26 June 2005, pp. 131–131. 13

[33] N. Ratha and A. Jain, "FPGA-based computing in computer vision," in *Computer Architecture for Machine Perception, 1997. CAMP '97. Proceedings Fourth IEEE International Workshop on*, 20-22 Oct. 1997, pp. 128–137. 13

[34] Z. Wei, D. Lee, and B. Nelson, "A hardware-friendly adaptive tensor based optical flow algorithm," *Lecture Notes in Computer Science*, vol. 4842, p. 43, 2007. 29

[35] C. Harris and M. Stephens, "A combined corner and edge detector," *Alvey Vision Conference*, vol. 15, 1988. 29

[36] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An Invitation to 3-D Vision: From Images to Geometric Models*, 1st ed., ser. Interdisciplinary Applied Mathematics. Springer, 2004, vol. 26. 29

[37] D. L. Johansen, "Video stabilization and object localization using feature tracking with small UAV video," Master's thesis, Brigham Young University, 2006. 30, 31, 43

[38] C. Schmid, R. Mohr, and C. Bauckhage, "Evaluation of interest point detectors," *International Journal of Computer Vision*, vol. 37, no. 2, pp. 151–172, 2000. 30

[39] P. Tissainayagam and D. Suter, "Assessing the performance of corner detectors for point feature tracking applications," *Image and Vision Computing*, vol. 22, no. 8, pp. 663–679, Aug. 2004. 31

[40] C.-H. Oertel, "Machine vision-based sensing for helicopter flight control," *Robotica*, vol. 18, no. 3, pp. 299–303, 2000. 31

[41] Y. Ninomiya, S. Matsuda, M. Ohta, Y. Harata, and T. Suzuki, "A real-time vision for intelligent vehicles," in *Intelligent Vehicles '95 Symposium., Proceedings of the*, 25-26 Sept. 1995, pp. 315–320. 31

[42] Y. Matsumoto and A. Zelinsky, "An algorithm for real-time stereo vision implementation of head pose and gaze direction measurement," in *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, 2000, pp. 499–504. 31

[43] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. New York, NY, USA: Cambridge University Press, 2000. 43

[44] P. Perez and N. Garcia, "Robust and accurate registration of images with unknown relative orientation and exposure," in *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, vol. 3, 2005, pp. III–1104–7. 43

[45] S. Mallick, "Feature based image mosaicing." 43

[46] S. Thunuguntla and B. Gunturk, "Feature-based image registration in log-polar domain," in *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, vol. 2, March 18-23, 2005, pp. 853–856. 43

[47] Y. Kanazawa and K. Kanatani, "Robust image matching under a large disparity," *Workshop on Science of Computer Vision*, pp. 46–52, 2002. 43

[48] J. Akita, *Real-Time Color Detection System Using Custom LSI for High-Speed Machine Vision.* London, UK: Springer-Verlag, 2000. 49

[49] D. Travis, *Effective Color Displays: Theory and Practice.* Academic Press London, 1991. 50

[50] Newton Labs, "The cognachrome vision system," 1999. [Online]. Available: http://www.newtonlabs.com/cognachrome/ 50

[51] J. Bruce, T. Balch, and M. Veloso, "Fast and inexpensive color image segmentation for interactive robots," in *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, vol. 3, 31 Oct.-5 Nov. 2000, pp. 2061–2066. 50

[52] J. Heikkila and O. Silven, "A four-step camera calibration procedure with implicit image correction," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, 17-19 June 1997, pp. 1106–1112. 50

[53] J.-Y. Bouguet, "Camera calibration toolbox for matlab," 2007. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/ 50

[54] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," *ACM SIGGRAPH 2001 Course Notes*, 2001. 50

[55] C. Zhang, P. Wong, and O. Selinus, "A comparison of outlier detection methods: exemplified with an environmental geochemical dataset," in *Neural Information Processing, 1999. Proceedings. ICONIP '99. 6th International Conference on*, vol. 1, 16-20 Nov. 1999, pp. 183–187. 51