Vision-Based Control of a Multi-Rotor Helicopter

Justin Haines

CMU-RI-TR-12-02

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

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

December 2011

Masters Committee:
Sanjiv Singh
Stephen T. Nuske
Michael Dille

Abstract

Autonomous air vehicles frequently rely on GPS as a primary source of state feedback. However, relying on GPS disallows operation in enclosed spaces, under heavy vegetation or near large obstacles since GPS does not provide sufficient accuracy in these environments. Similar work in GPS-denied navigation uses laser-based odometry, structured-light or places visual markers in the environment. These approaches are not appropriate for operating in unstructured environments, therefore it is necessary to develop a system around more robust vision-based techniques.

This work presents a method for controlling an autonomous, multi-rotor helicopter based on visual odometry which consists of two major sections. First, a method for developing an accurate dynamic model of a multi-rotor helicopter based on a combination of first priciples and emperical data is presented. This modeling technique is used to develop a control system which enables trajectory tracking. Second, a vision-based state-estimation technique is presented along with a hardware implementation that enables execution of the algorithms in real-time on board a small vehicle with strict payload constraints. The methods described are implemented in flight-ready hardware with a minimal weight, power and computational footprint. The system is then evaluated on board a small, eight-rotor helicopter. We successfully demonstrate vision-based trajectory tracking in flight on this vehicle.

Contents

List of Figures

List of Tables

1 Introduction

There are many compelling applications for small unmanned aerial vehicles (UAV). These applications include mapping, surveillance, exploration and others. Many of these domains require the vehicle to be able to execute missions through unstructured environments and be able to operate close to obstacles. These types of missions are more easily carried out by a smaller vehicle since it will be able to fit through small openings and more reliably be able to operate near obstacles safely. It is for these reasons that this work is targeted at vehicles no more than 1 meter in diameter and no more than 3.0 kg in mass. This small vehicle size limits the payload capacity to less than 1.0 kg, so weight is a strong consideration when designing the sensors and computation for the vehicle. This work presents a computing architecture that enables a higher performance to weight ratio than previously possible for UAVs.

In order to allow a vehicle to explore an environment completely, it is attractive to allow the vehicle to operate inside buildings, near large obstacles, or under heavy vegetation. Unfortunately, these environments prevent the vehicle from receiving a strong, reliable GPS signal. Therefore, we need to develop a method for performing state estimation without the availability of GPS. This work presents a vision-based solution to the state estimation problem which can provide the necessary feedback to stabilize the vehicle's flight.

In order to allow such a vehicle to carry out a mapping or exploration mission, it must be able to operate over a long distance. The longer distance that the vehicle can cover before needing to recharge its batteries, the more area it will be able to explore. Due to the unreliable and non-deterministic nature of wireless links, we cannot assume it is possible to communicate with the vehicle over these large distances. Therefore, this work presents a hardware architecture that allows for all of the computation necessary to perform its mission to be on board.

This work presents a software and hardware solution to the trajectory tracking problem. The solution presented is appropriate for enabling trajectory control on a small, multi-rotor helicopter and allows for all of the computation to be performed on board. Section 3 introduces a dynamic model of a multi-rotor vehicle. This section discusses a method for determining the physical parameters necessary for creating a simulation of a multi-rotor helicopter. Section 4 discusses the control techniques implemented to stabilize the vehicle's velocity and enable trajectory tracking. Sections 5 and 6 discuss the vision-based state estimation system and the hardware implementation that enables real-time execution of the algorithms on board the vehicle. Sections 7 and 8 describe the vehicle on which these algorithms were tested and the results of the flight experiments.

2 Related Work

There has been a significant amount of work in the area of GPS-denied navigation for multi-rotor vehicles. The state-estimation systems for these vehicles falls into three broad categories: vision-based, laser-based and structured-light-based. The computation for these systems is done on-board the vehicle in some cases, and is done off-board

in others. The controllers used to stabilize these systems are typically either PID or LQR type controllers.

Structured light is an excellent approach for indoor helicopter navigation since the low-cost Kinect sensor is available that produces depth images directly. Huang et. al. [1] present a method for using a Kinect for creating a map of an indoor environment, but this technique will not work outdoors. This is because the brightness of an outdoor environment prevents the sensor from working correctly. Also, this type of sensor only works as short range, so the vehicle will not be able to operate away from obstacles. This group implements a PID type position controller, which is very similar to the technique presented in this work, and they are able to achieve satisfactory position tracking.

Laser-based state-estimation is a good approach to this type of problem because it yields accurate positioning results and is not as computationally demanding as vision-based solutions. Achtelik et. al. [2] present a laser-based localization system for use on a small UAV. While this type of system will work well in structured environments with distinctive 3-D structures near the vehicle, it will not work well outside of these assumptions. A critical failure of this system is that it is limited by the range of the LIDAR. If there is no structure within the sensing range of the laser, the vehicle's motion cannot be calculated. The technique presented in [2] implements a LQR type controller to stabilize the flight of the vehicle. This control technique is applicable because they have also implemented an Extended Kalman Filter which provides full state feedback. Without this filter, the latency of the state estimation system disallows an LQR control scheme. Section 10 discusses the application of this approach to the system presented in this work.

There are several groups (Minh and Ha [3]; Meier et. al. [4]; Wu, Johnson and Proctor [5]) who have presented work using cameras with fiducials in the environment. This technique reduces the computational complexity of the vision algorithms significantly which enables the algorithms to be executed on-board the vehicle in real-time. However, this technique is not extensible to a large environment since it is impractical to place markers throughout a large region.

Other groups handle the computational complexity of vision-based algorithms by performing the computation on heavy, powerful, ground-based computers. The vehicle must send the images to the ground computer which executes the algorithms and sends back control commands. Achtelik et. al. [2] and Blösch et. al. [6] both present techniques that require an off-board computer. This solution to the computation problem is not reasonable for vehicles that will travel a long distance because of the unreliability of wireless links. These wireless systems have non-deterministic latency which is unacceptable for stabilizing a vehicle in flight. Conte and Doherty [7] solve the computation problem by using a large vehicle. They use a Yamaha R-Max vehicle which has a total weight of close to 100 kg. This vehicle has a large enough payload capacity that it can carry heavy, powerful computers on board. This type of vehicle is good for flying far away from obstacles, but its size prevents it from operating in confined spaces.

The most similar system to the one presented in this work is described by Voigt et. al. in [8]. This vehicle uses a vision-based system for localization and performs all of the computation on-board a small vehicle. This general approach meets the requirements of an outdoor vehicle that will carry out long missions, and is very similar

Table 1: Dynamics Variables

Symbol	Definition
ϕ	roll angle
θ	pitch angle
ψ	yaw angle
Ω	total rotor speed
$I_{x,y,z}$	body inertia
J_r	rotor inertia
U_1	total thrust
U_2	left/right thrust differential
U_3	front/back thrust differential
U_4	cw/ccw torque differential

to the techniques presented in this work. The limitation of the system described in [8] is that all of the on-board computation power is dedicated to the vision system, leaving no available computation power for other algorithms. In order for a vehicle to carry out a meaningful mission, it must have available computation power for planning, mapping and other tasks. This problem is addressed in Section 6 which introduces a novel computing architecture which allows the execution of computationally expensive vision algorithms without consuming all of the computing resources available on the vehicle.

In contrast to the existing work in GPS-denied navigation, this work presents a vision-based system that is capable of operating in unstructured environments while performing all of the required computation on-board a small vehicle. This system enables operation of a small vehicle in a wide variety of GPS-denied environments while leaving computational resources available for the other tasks required to carry out a meaningful mission.

3 Dynamic Simulation

In order to develop a trajectory tracking controller, it is first necessary to develop a dynamic model of the system to be controlled. This is useful for determining an appropriate control approach and is also useful for testing the controller in simulation. By developing an accurate model of the vehicle, control algorithms can be tested in simulation before they are tested in flight. This increases the ability to iterate the control design quickly without requiring time-consuming flight tests that may risk a crash.

The equations of motion for a multi-rotor helicopter, as described in [9] and [10], are shown in Eq. 1 with the notation shown in Table 1. These equations are derived from first principles and are applicable to a general multi-rotor helicopter. They neglect any aerodynamic effects and do not include any external disturbances.

$$\begin{cases} \ddot{x} & = (cos\phi sin\theta cos\psi + sin\phi sin\psi)\frac{1}{m}U_1 \\ \ddot{y} & = (cos\phi sin\theta sin\psi - sin\phi cos\psi)\frac{1}{m}U_1 \\ \ddot{z} & = -g + (cos\phi cos\theta)\frac{1}{m}U_1 \\ \ddot{\phi} & = \dot{\theta}\dot{\psi}(\frac{I_y - I_z}{I_x}) - \frac{J_r}{I_x}\dot{\theta}\Omega + \frac{l}{I_x}U_2 \\ \ddot{\theta} & = \dot{\phi}\dot{\psi}(\frac{I_z - I_x}{I_y}) + \frac{J_r}{I_y}\dot{\phi}\Omega + \frac{l}{I_y}U_3 \\ \ddot{\psi} & = \dot{\phi}\dot{\theta}(\frac{I_x - I_y}{I_z}) + \frac{1}{I_z}U_4 \end{cases} \tag{1}$$

From these equations, it can be seen that the linear dynamics depend upon the vehicle's attitude, the total thrust produced by the vehicle and the vehicle's mass. The rotational dynamics depend on the moments of inertia of the vehicle and rotors as well as the rotor speeds and torques, but do not depend upon the linear state of the vehicle.

Since the rotational dynamics can be considered independently of the linear dynamics, we will first develop a model that describes the state evolution for the rotational subsystem. Since this work is designed to be implemented on an off-the-shelf helicopter, we assume that the vehicle contains an existing attitude controller. This assumption further complicates the rotational dynamics of the system, since we do not know the dynamics of the attitude controller. This means that the attitude dynamics are dependent on several unknown parameters including the physical properties of the vehicle and the unknown dynamics of the low-level control system.

3.1 Attitude Modeling

The complex nature of the attitude response indicates that modeling the system directly will be prohibitively difficult since it depends on too many unknowns. In order to simplify the system, we make some assumptions. The first assumption is that the roll, pitch and yaw dynamics are independent. This independence assumption ignores the gyroscopic terms that appear in Eq. 1, but the low-level control system will be attempting to minimize these effects, so it is acceptable to neglect them. This assumption turns the coupled attitude sub-system in Eq. 2 into the decoupled attitude system in Eq. 3 with the notation shown in Table 2.

$$\begin{bmatrix} \phi \\ \theta \\ \dot{\psi} \end{bmatrix}_t = f(\begin{bmatrix} \phi^d \\ \theta^d \\ \dot{\psi}^d \end{bmatrix}_{0:t}) \tag{2}$$

$$\begin{cases} \phi_t & = f_\phi(\phi^d_{0:t}) \\ \theta_t & = f_\theta(\theta^d_{0:t}) \\ \dot{\psi}_t & = f_{\dot{\psi}}(\dot{\psi}^d_{0:t}) \end{cases} \tag{3}$$

Table 2: Attitude Model Variables

Symbol	Definition
ϕ^d	desired roll angle
θ^d	desired pitch angle
$\dot{\psi}^d$	desired yaw rate

The second assumption made to enable attitude modeling is that f_θ and f_ϕ can be represented as two-dimensional state-space systems and $f_{\dot{\psi}}$ can be represented as a one-dimensional state-space system. This assumption further simplifies the rotational dynamics to the form shown in Eq. 4. By presuming a state-space model of the dynamics, linearity in the attitude response is assumed. This assumption is not valid for extreme attitudes, but remains a good approximation near hover.

$$\begin{cases} \dot{x_\phi} &= A_\phi * x_\phi + B_\phi * \phi^d \\ \phi &= C * x_\phi \\ \dot{x_\theta} &= A_\theta * x_\theta + B_\theta * \theta^d \\ \theta &= C * x_\theta \\ \dot{x}_{\dot{\psi}} &= A_{\dot{\psi}} * x_{\dot{\psi}} + B_{\dot{\psi}} * \dot{\psi}^d \\ \dot{\psi} &= C * x_{\dot{\psi}} \end{cases} \tag{4}$$

In order to develop a model to be used for simulating the vehicle, we must determine all of the unknowns in Equation 4. These parameters encapsulate all of the unknowns in the attitude sub-system, so they include effects from the physical characteristics of the vehicle and the dynamics of the attitude control system embedded in the vehicle. An experimental system identification approach is used to determine these parameters.

With this simplified model of the attitude response it is possible to apply the model identification techniques described by Ljung in [11]. These techniques require that the input and output of each system can be observed. By applying a known command to the attitude controller on the vehicle and observing the attitude response of the vehicle, we can apply these techniques to determine the parameters to the models in Eq. 4. This experiment was carried out on board a vehicle described in Section 7 with an IMU used to measure the attitude of the vehicle at 50 Hz.

Figure 1 shows the pitch command applied to the vehicle (red) along with the observed response (blue). The green curve shows the predicted response from the model that was fit to the data. It can be seen that the predicted response closely represents the true pitch response. Over this command sequence, the RMS model prediction error was 1.42 degrees.

Figure 2 shows the same information for the yaw rate command, observed response and modeled response. Over this command sequence, the RMS yaw rate prediction error was 1.9 degrees per second.

Due to the linearity assumption made in determining the form of the attitude dy-

namics, the large magnitude commands carry most of the error in the prediction. This is because at these points, the linearity assumption breaks down. In practice, the vehicle's attitude will not deviate more than five degrees from horizontal, so these effects will not be noticed.

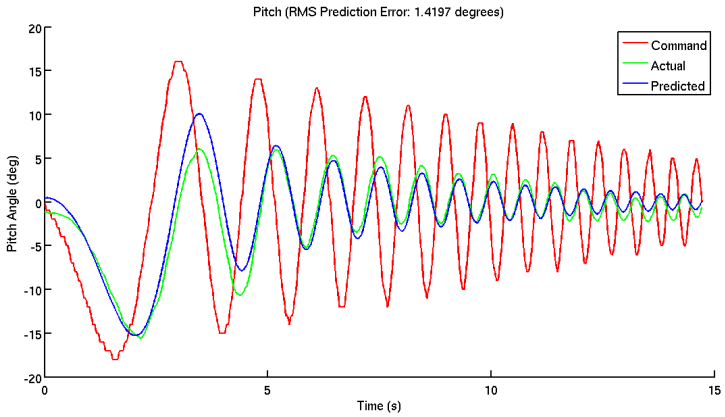

Figure 1: Pitch Response Modeling

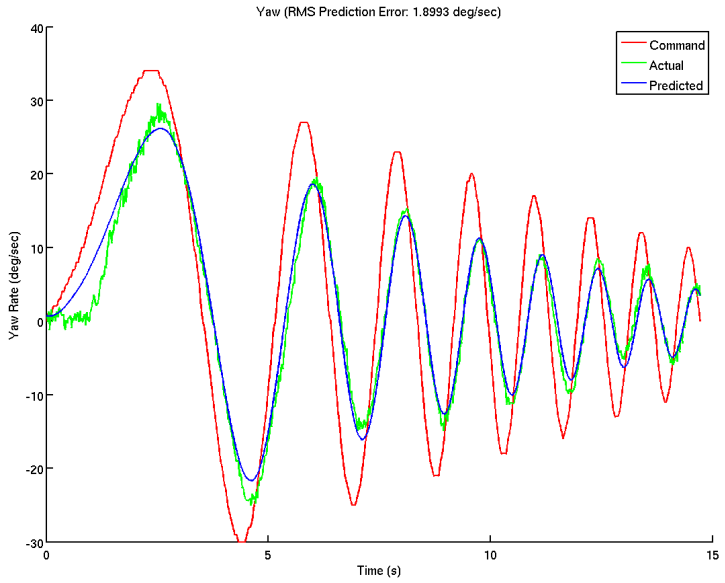

Figure 2: Yaw Response Modeling

3.2 Velocity and Position Modeling

As shown in Eq. 1, the linear dynamics of a vehicle of this type depend on the attitude of the vehicle (which can be measured by an IMU), the mass of the vehicle (which can also be measured) and the total thrust produced by the vehicle. The motor dynamics are assumed to be much faster than the translational dynamics, so we can assume that the thrust produced by the vehicle is a direct function of the command issued to the vehicle with some delay, t_d, to account for communication latency. This function, shown in Eq. 5, will be referred to as the vehicle's thrust curve.

$$T_N(t) = f_T(T_d(t - t_d)) \tag{5}$$

Since the vehicle will usually be near a hover, we will linearize this function about a hover from experimental data. A simple method for determining this curve is manually hovering the vehicle while varying the weight of the payload. By measuring the command necessary to maintain a hover, it is possible to draw samples from the thrust curve.

Now that all of the parameters of the linear dynamics are known, we can apply simple Euler integration to the second order differential equations describing the linear motion in Eq. 1.

3.3 State Estimation Modeling

In addition to modeling the dynamic response of the vehicle, it is useful to model the dynamics of the state estimation system. As will be discussed later, the dynamics of the state estimation play a significant role in the closed-loop dynamics of the controlled system, so it is beneficial to include them in a dynamic simulation.

The general form of the state estimation dynamics for a single state variable, x_i, is shown in Eq. 6. A typical state-estimator will sample the desired state variable at some period, dt, may average that variable over a window of time (t_s) and will introduce some latency in its estimate, t_d. The resulting form of a state estimation function is shown in Eq. 7.

$$\hat{x_i}(t) = f(x_i(0:t)) \tag{6}$$

$$\hat{x_i}(t) = \frac{1}{t_s} \int_{\bar{t}-t_d-t_s}^{\bar{t}-t_d} x_i(\tau) d\tau, \bar{t} = \lfloor \frac{t}{dt} \rfloor * dt \tag{7}$$

This form of the state estimator is able to capture the elements of the observation dynamics that arise because of the sampling averaging and latency, but do not capture the noise of the estimation itself. If the specific technique of state estimation produces noisy samples of the true state, this noise must also be taken under consideration in the dynamic model of the state estimator.

4 Control Architecture

As mentioned in Section 3, the vehicle is assumed to have an attitude controller embedded in it that will take as commands a desired roll, pitch, yaw rate and total thrust. A cascaded control approach is taken where the inner control loop is a velocity controller that accepts a 3-dimensional velocity command and produces commands applied to the existing low-level attitude controller. The outer loop of the control system achieves trajectory tracking. This loop receives a desired path from the planner and issues commands to the velocity controller to track the path. Figure 3 shows the overall system diagram.

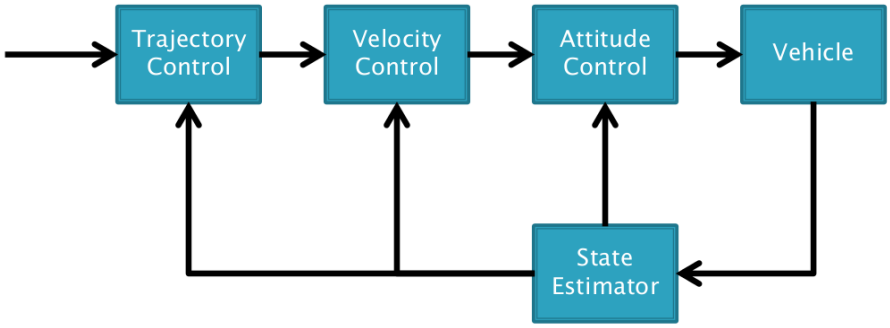

Figure 3: Overall Control System Architecture

4.1 Velocity Control

The velocity controller receives an x, y, z velocity command and issues roll, pitch and thrust commands to try to achieve the desired velocity. The velocity controller is also responsible for controlling the vehicle's heading. To do this, it receives a heading command and issues a heading rate to the vehicle. On a multi-rotor vehicle as described in Section 3, the heading can be decoupled from the translational dynamics, so it will be controlled independently.

The horizontal velocity is controlled by varying the roll and pitch of the vehicle to direct the thrust vector of the vehicle in the desired direction of travel. The vertical velocity is controlled by varying the total downward thrust of the vehicle.

This work presents a control system based on PID controllers. PID is selected primarily because of its simplicity and robustness. Other, more advanced, techniques such as LQR or backstepping are more sensitive to modeling errors and less tolerant of state-estimation latency. These techniques can potentially yield improved performance, but PID is able to provide satisfactory performance for the system discussed below.

Table 3 shows the symbols used to develop the velocity control equations.

The velocity error is calculated according to Eq. 8.

Table 3: Velocity Control Variables

Symbol	Definition
m	vehicle mass
g	gravity
u^x	x-axis aligned command
u^y	y-axis aligned command
u^ϕ	roll angle command
u^θ	pitch angle command
$u^{\dot{\psi}}$	yaw rate command
T_z	desired vertical thrust
T	total desired thrust
$f_T(\cdot)$	vehicle's thrust curve
u^T	total thrust command
R_ψ	yaw rotation matrix
$v^d_{x,y,z}$	desired velocity
ψ^d	desired yaw angle
$v_{x,y,z}$	current velocity
ψ	current yaw angle
$e^v_{x,y,z}$	velocity error
K_{Ph}	horizontal velocity proportional gain
K_{Ih}	horizontal velocity integral gain
K_{Dh}	horizontal velocity derivative gain
K_{Pv}	vertical velocity proportional gain
K_{Iv}	vertical velocity integral gain
K_{Dv}	vertical velocity derivative gain
$K_{P\psi}$	yaw angle proportional gain

$$\begin{bmatrix} e^v_x \\ e^v_y \\ e^v_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} - \begin{bmatrix} v^d_x \\ v^d_y \\ v^d_z \end{bmatrix} \tag{8}$$

After the velocity error is computed, a PID controller is applied to the horizontal velocity error according to Eq. 9. The resulting commands are rotated into the body frame of the vehicle and applied as roll and pitch commands according to Eq. 10

$$\begin{cases} u^x = K_{Ph} e^v_x + K_{Ih} \int_0^t e^v_x + K_{Dh} \dot{e^v_x} \\ u^y = K_{Ph} e^v_y + K_{Ih} \int_0^t e^v_y + K_{Dh} \dot{e^v_y} \end{cases} \tag{9}$$

$$\begin{bmatrix} u^\phi \\ u^\theta \end{bmatrix} = R_\psi \begin{bmatrix} u^x \\ u^y \end{bmatrix} \tag{10}$$

The thrust command is computed from the vertical velocity error, e^v_z, according to Eq. 11. First a PID controller is applied to the velocity error to compute the desired

vertical thrust, then the total thrust is found that has the desired vertical component. Finally, the thrust command is determined by applying the inverse thrust curve to the desired total thrust.

$$\begin{cases} T_z = mg + K_{Pv}e_z^v + K_{Iv}\int_0^t e_z^v dt + K_{Dv}\dot{e_z^v} \\ T = \frac{T_z}{cos(\phi)cos(\theta)} \\ u^T = f_T^{-1}(T) \end{cases} \tag{11}$$

As mentioned earlier, the heading of the vehicle is controlled independently from the velocity. The yaw rate command is developed according to a simple proportional controller as shown in Eq. 12

$$u^{\dot{\psi}} = K_{P\psi}(\psi^d - \psi) \tag{12}$$

Figure 4 shows the simulated and actual vehicle response to a horizontal velocity command of 3 m/s. Figure 5 shows the simulated and actual vehicle response to a vertical velocity command of 1 m/s. The observed velocity curve is determined from the on-board state-estimation system described in Section 5.

It is clear from these plots that the simulation of the vehicle represents the true response closely. The noise on the observed velocity curves is due to noise in the state-estimation system and was not present in the actual vehicle's motion. This type of noise is very hard to model, so it is not included in the vehicle simulation.

It is also clear from these plots that the velocity response is fairly slow; the vehicle takes about 2 seconds to achieve the desired velocity. This is primarily due to the speed of the attitude subsystem, which cannot be changed. The roll and pitch dynamics have a 2 percent settling time of 1.87 seconds. This slow response limits the performance of the velocity controller. Another factor which limits the velocity tracking performance is the state-estimation dynamics. The state-estimation (discussed further in Section 5) has a measurement latency of about 130ms. This delay means that the velocity controller is likely to become unstable if it is tuned more aggressively to try to improve performance.

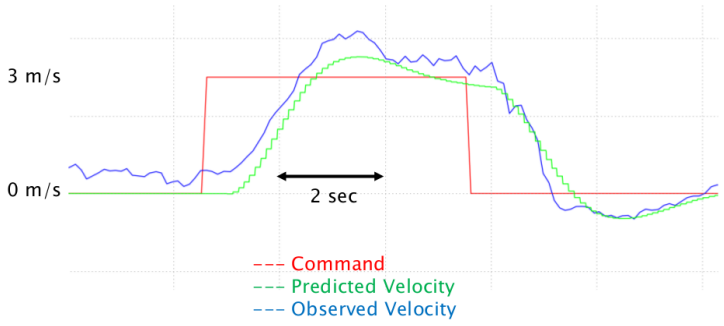

Figure 4: Horizontal Velocity Tracking Performance

Figure 5: Vertical Velocity Tracking Performance

4.2 Trajectory Control

After successfully stabilizing the velocity of the vehicle, it is possible to implement a trajectory controller. This controller will receive trajectories from the planner and issue velocity commands to the velocity controller in order to track the trajectory. The algorithm presented is based on the work described by Hoffmann, Waslander and Tomlin in [12]. Figure 6 and Table 4 show the various quantities used to develop the trajectory control algorithm.

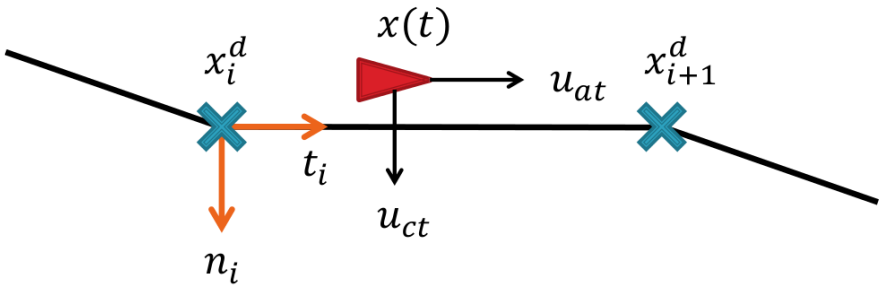

Figure 6: Trajectory Control

The cross-track error is defined as the distance from the current position to the closest point along the path and the derivative of this error si the magnitude of the velocity in the same direction. The along-track velocity error is defined as the the difference between the velocity encoded in the path and the velocity of the vehicle tangent to the path. Eq. 13 shows the formulation of these error terms.

Table 4: Trajectory Control Variables

Symbol	Definition
$x(t)$	current vehicle position
$v(t)$	current vehicle velocity
x_i^d	trajectory waypoints
s_i^d	desired vehicle speed
t_i	unit tangent to trajectory segment i
n_i	unit normal to trajectory segment i
e_{ct}	cross-track position error
$\dot{e_{ct}}$	cross-track velocity error
$\dot{e_{at}}$	along-track velocity error
u_ct	cross-track velocity command
u_at	along-track velocity command
u	total velocity command
ψ_i^d	desired yaw angle
K_{Pct}	cross-track proportional gain
K_{Ict}	cross-track integral gain
K_{Dct}	cross-track derivative gain
K_{Pat}	along-track proportional gain
K_{Iat}	along-track integral gain

$$\begin{cases} e_{ct} = (x_i^d - x(t)) \cdot n_i \\ \dot{e_{ct}} = -v(t) \cdot n_i \\ \dot{eat} = v_i^d - v(t) \cdot t_i \end{cases} \tag{13}$$

The cross-track velocity command consists of a PID controller wrapped around the cross-track position error. Eq. 14 shows the formulation of the cross-track velocity command.

$$u_{ct} = K_{Pct} e_{ct} + K_{Ict} \int_0^t e_{ct} dt + K_{Dct} \dot{e_{ct}} \tag{14}$$

The along-track velocity command consists of a PI controller wrapped around the along-track velocity error. Eq. 15 shows the formulation of the along-track velocity command.

$$u_{at} = K_{Pat} \dot{e_{at}} + K_{Iat} \int_0^t \dot{e_{at}} dt \tag{15}$$

The total velocity command applied to the vehicle, u, is the sum of the along-track and cross-track velocity commands.

Figure 7 shows the simulated trajectory tracking performance of the algorithm described above. It can be seen that the vehicle's realized trajectory (indicated by the red

arrows) does not achieve the desired trajectory (shown in blue). This is because of the phase lag in the closed loop dynamics of the velocity controller. Since the commanded velocity is based upon the tangent of the desired path at the closest point, by the time the vehicle reaches that velocity it will already be further along the path and be required to change velocity. An additional problem that must be considered is the overshoot at the end of the path. When the vehicle reaches the 12 o'clock position, the trajectory ends and it is commanded to hover at the last waypoint in the trajectory. The vehicle attempts to hold a constant velocity along the entire path, so when it reaches the end, it will overshoot while attempting to slow down.

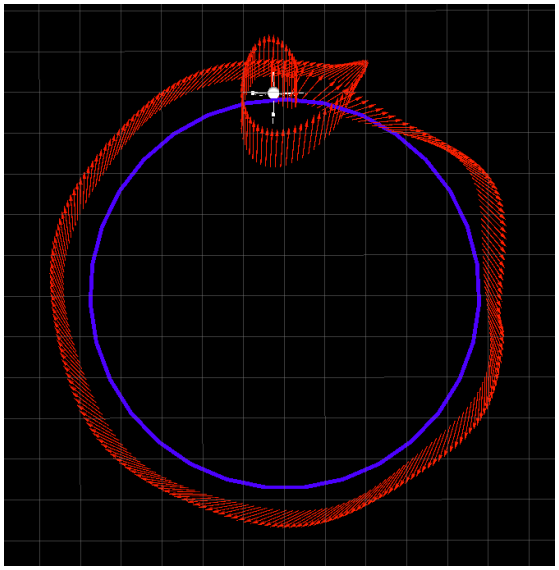

Figure 7: Simulated Loop Trajectory Tracking Performance (1 meter grid)

These problems can be addressed by basing the commanded velocity on the desired velocity at a future point in the trajectory rather than the closest point. This is implemented by finding the closest point to the vehicle along the path and traversing the trajectory a distance which is a constant multiple of the current vehicle speed. By making this distance a function of speed, the vehicle will consider more distant future velocities when it is moving more quickly. Figure 8 shows the simulated trajectory tracking performance with this modification to the algorithm. In this figure, the vehicle was traveling at approximately 1.5 m/s which results in a 1.5 meter lookahead distance.

5 State Estimation

Section 4 describes a control approach which can be applied to stabilize a helicopter in flight, but it assumes that the vehicle's state can be measured. Section 3 considers the dynamics of a state-estimator, but do not discuss the implementation of such a estimator.

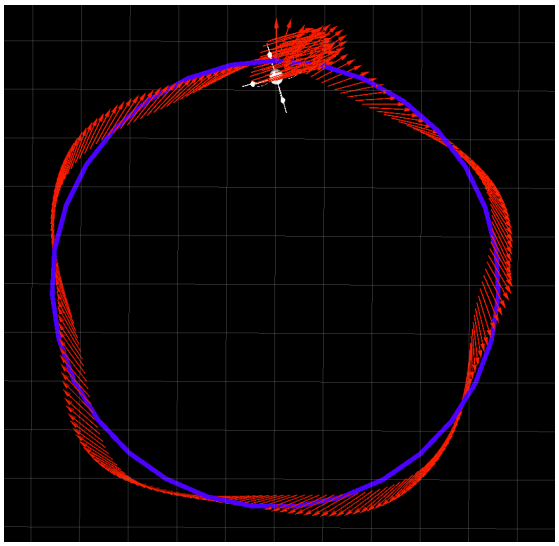

Figure 8: Simulated Loop Trajectory Tracking Performance (1 meter grid)

There are several things that must be considered in designing a state-estimation system that will provide feedback for the controllers described in Section 4. The selected method must not require any heavy sensors, the associated algorithms must be able to run in real-time on board the vehicle and it must be robust. Visual odometry is selected as the primary source of state feedback for this vehicle. It is a good choice since the cameras required can be very light and it functions well in an outdoor environment. Vision-based solutions are not ideal since they are often very computationally expensive. This problem will be addressed later in Section 6.

5.1 Visual Odometry

The specific algorithm used for visual odometry is described in [13]. It is a stereo-based, frame-to-frame approach. The algorithm extracts corner-like features from two consecutive stereo image pairs. It then matches these features across all four images using gradient block-matching. The matched features are filtered through a spatial bucketing technique which removes some matches in an attempt to distribute the matches evenly over the field of view. Following this, the matches are put through a RANSAC-based outlier rejection process. The resulting inliers are used to calculate the rotation and translation of the vehicle over the time between consecutive frames.

Since visual odometry is a frame-to-frame technique, it has the inherent drawback of allowing the state estimate of the vehicle to drift over time. While this problem is unavoidable in this type of algorithm, the state estimation system will not be useful if this drift rate is too high. Figure 9 shows the vehicle's pose as calculated by the visual odometry algorithm. The data for this experiment was captured on a helicopter in flight which takes off and lands at the same point. The images used were 320 by 256 pixels

and were captured at 10 frames per second. This flight took 101 seconds, resulting in over 1000 frame-to-frame transformations to be calculated by the visual odometry algorithm. The path shown is 130 meters in length and the accumulated error over this distance is 1.45 meters. We assume that the planner will be re-planning regularly, so this drift rate of about 1cm per meter is acceptable for trajectory control of the vehicle.

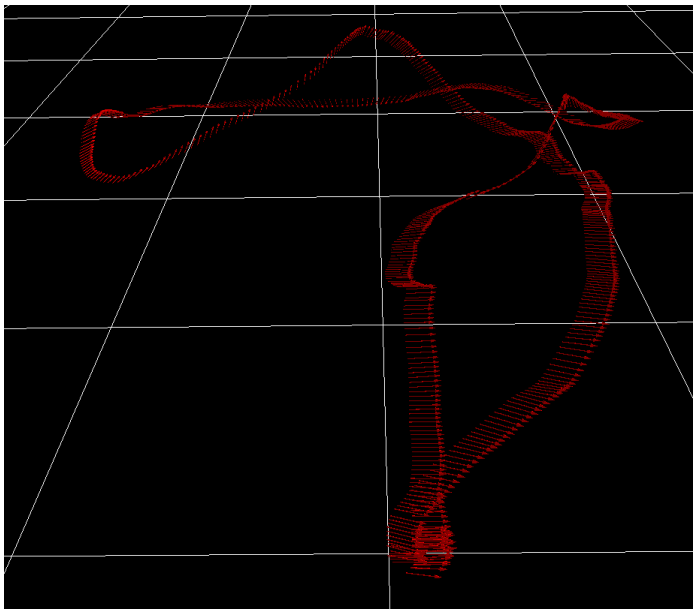

Figure 9: Pose in Flight Calculated by Visual Odometry (10 meter grid)

6 Embedded Vision Processing

As mentioned in Section 5, the vision-based state-estimation algorithm selected is computationally expensive. This algorithm must execute in real-time, so it must have a significant amount of computational resources allocated to it. Even though the state-estimation algorithms are very important, it is also important that there is enough computational power available to the other algorithms on the vehicle so that it can carry out its mission. It would be possible to run the vision-processing algorithms on a single computer on board the vehicle, but because of the payload weight limitations, that computer would have to allocate most of its resources to the vision processing.

Figure 10 shows the break-down of the computation time for the various parts of the algorithm. The first step (”Capture Frame” in light blue) is the time it takes for the camera to acquire a frame and transmit it to the computer. The next step (”Debayer, Rectify, Undistort” in red) is where the calibration parameters are applied to the images to convert them to the form that the vision algorithms can use. The next two steps (”Extract Features” in orange and ”Match Features” in dark blue) make up the first

part of the visual odometry algorithm where the interest points are found and matched among the two sequential stereo pairs of images. The final section ("Visual Odometry" in purple) is the step where the outlier rejection and egomotion estimation actually take place. It can be seen that the bulk of the processing time is consumed on the generic vision tasks of pre-processing, feature extraction and feature matching while less than 1 percent of the time is spent calculating the motion of the cameras.

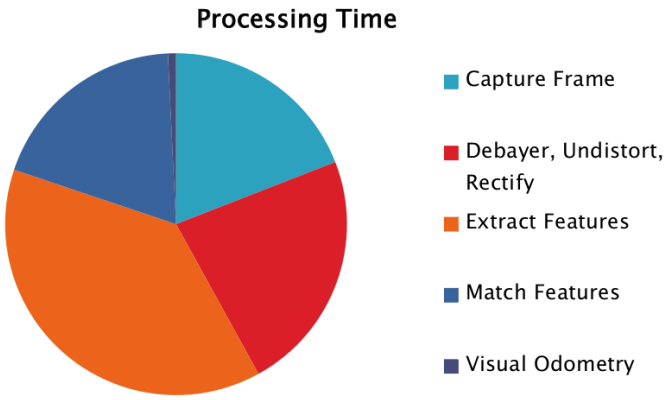

Figure 10: Computation Time of Algorithm Steps

Since the bulk of the processing time is spent on such a generic task, it would be beneficial to offload this processing onto a secondary computer that is designed specifically to handle the tasks of feature extraction and matching. Of course, this secondary computer must be very small, light and low-power. Fortunately, the cell-phone market has the same requirements for their computing platforms. For this reason, a cell-phone processor is selected to fill the role of the secondary computer. Figure 11 shows the Logic PD Torpedo Computer on Module (COM). This COM has a 1 GHz ARM Cortex-A8 processor with a TMS320C64x+ DSP co-processor. It also has 512 MB of flash memory and 256 MB of RAM. This entire package is just 1.96 grams and uses less than 1 watt of power.

This computing module is ideal for the intended application since it is so small and light. The DSP co-processor is also an excellent resource since it enables fast execution of the math-heavy algorithms such as convolution that normally consume much of the processing time in a feature extraction algorithm.

Unfortunately, a single module of this type is not capable of executing the required algorithms in real-time. For this reason, a parallel vision processing pipeline is introduced so that the total computation can be split between four of these computing modules. The parallelization scheme used is that each image to be processed is divided into four sub-images and each processor performs the feature extraction on each sub-image. After the feature extraction is complete, the processors share the interest points and then each matches one quarter of the feature points against the entire set. The result is an algorithm that is equivalent to the original one processor implementation, but divides the work among multiple processing units.

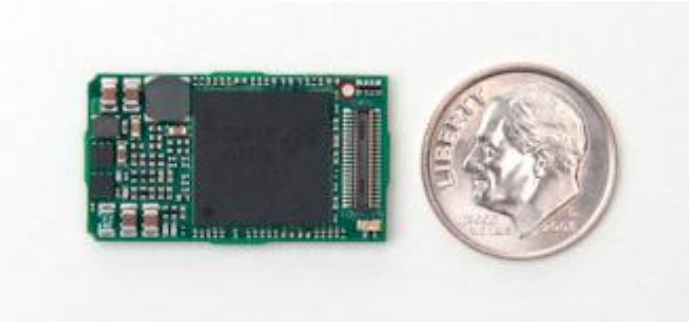

Figure 11: Torpedo Computer on Module

Figure 12 shows the custom carrier board used to connect to four of the Torpedos. The carrier board connects to the host computer over a single USB channel and has a USB hub that enables communication with all four of the Torpedos.

It is worth noting that using four processors does not divide the processing time by four, instead it reduces it by approximately one half. This is because the four processors share a USB channel to communicate with the host, so the distribution of images takes the same amount of time regardless of the number of processors used. An additional reason for this is that the sub-images must overlap by about 10 pixels since the interest point descriptors cannot be generated for interest points too close to the edge of the image. This results in the total number of pixels to be processed increasing by about 12 percent when moving from one processor to four.

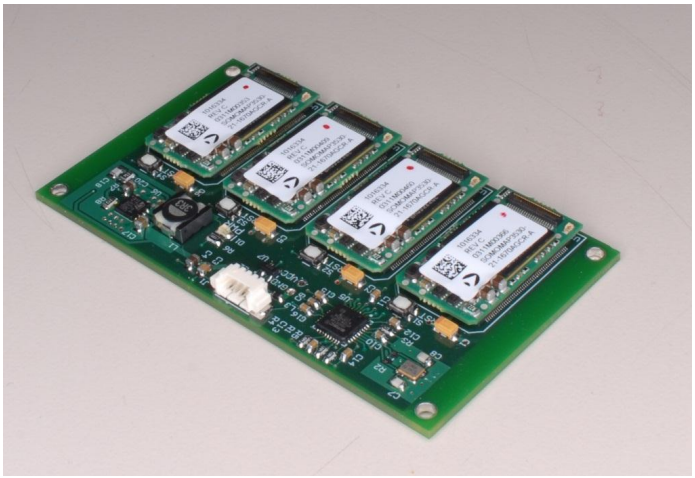

Figure 12: Custom Vision-Processing Computer

This custom-made vision computer is 23 grams total and uses less than 4 watts of power. The computer is able to execute the feature extraction and matching algorithms

on 320x256 images at 10 frames per second.

Figures 13 and 14 show the processing time of the visual odometry algorithm without and with the vision-processing computer. This profile was generated with a dual-core 1.86GHz main computer. The segments shown with cross-hatched colors are periods of time in which USB data transfer takes place. These segments take time due to the data transfer, but do not consume any significant processing resources. These figures show two significant results. First, the total processing time per frame increases slightly from 116 ms to 131 ms with the addition of the vision-processing computer. This is because of the additional time required to distribute the images to the secondary processing units. The second result shown here is that the processing time on the main computer is reduced from 91ms to 31 ms per frame with the addition of the vision-processing computer. If the images are processed at 10 Hz, this means that the processing load on the main computer is reduced from 91 percent to 31 percent. This difference represents a significant gain that allows the main computer to carry out many other tasks beyond just the vision processing algorithms. Without the secondary vision computer, the main computer would not have any available processing time to execute other algorithms required for the vehicle to carry out its mission.

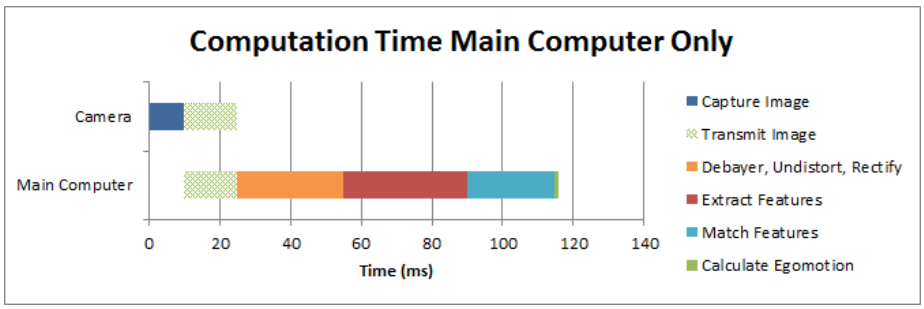

Figure 13: Processing Time Without Vision-Processing Computer

7 Experiments

7.1 Platform

The vehicle selected to test the algorithms described above is the Droidworx AD-8, an eight-rotor helicopter shown in Fig. 15. This vehicle was selected for two reasons. The first reason is that it has the longest flight time with the heaviest payload for a commercially available helicopter in this size class. The second reason is that an eight-rotor vehicle is capable of remaining in flight when up to two of the motors fail. This will prevent a crash in the event of a failed motor.

This vehicle has two 5500 mAh 4-cell lithium polymer batteries on board. These batteries provide the vehicle with a 20 minute flight time while carrying a 1.0 kg payload. The vehicle can carry up to 2.0 kg, but it reduces the flight time significantly.

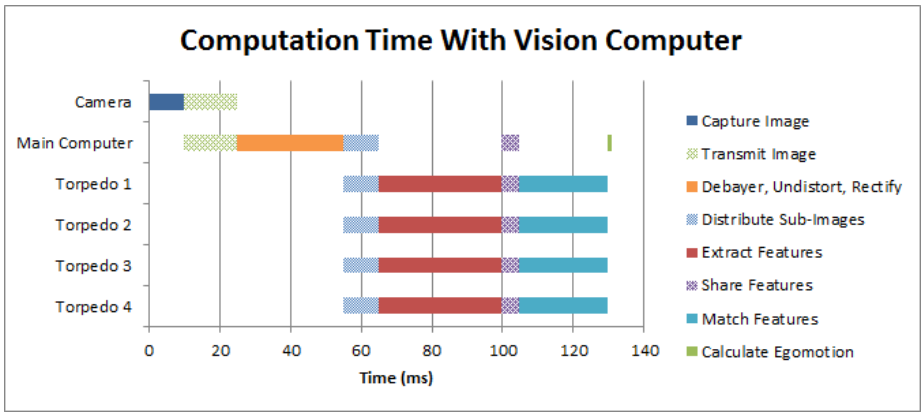

Figure 14: Processing Time With Vision-Processing Computer

The vehicle is controlled via a bi-directional serial port. The main computer sends commands at 25 Hz to the vehicle that contain the desired roll angle, pitch angle, yaw rate and total thrust. The vehicle sends telemetry data back to the main computer at 10 Hz which includes the vehicle's current attitude, motor velocities, and battery voltage.

Figure 15: Droidworx AD-8 Octorotor

7.2 Sensor Payload

There are two primary sensors on board the vehicle: a stereo camera rig, and a MEMS INS. Figure 16 shows the cameras and INS mounted on the vehicle payload. The

cameras and INS are rigidly mounted to a carbon fiber tube. This entire assembly is mounted on vibration dampers to prevent the rotor vibration from being transferred to the INS.

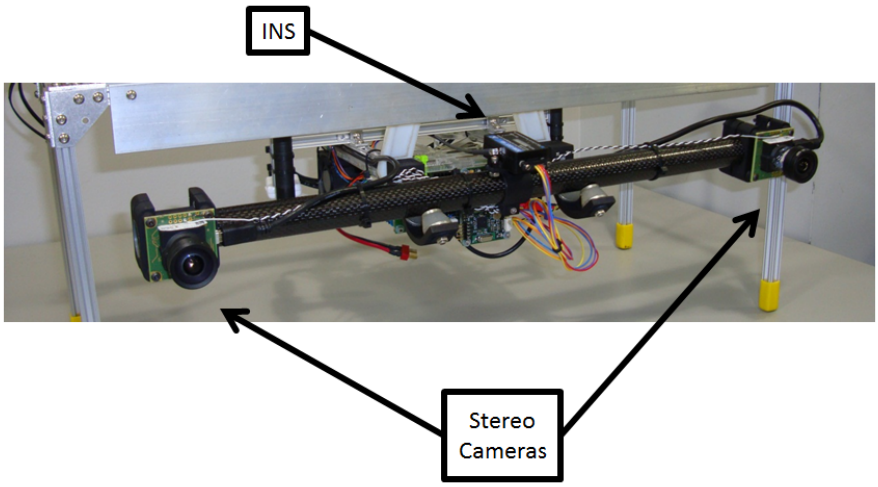

Figure 16: Vehicle Payload with Cameras and INS

The cameras are computer-vision grade, global-shutter, high dynamic range CMOS devices. They have a maximum resolution of 1280 x 1024 and maximum frame rate of 25 frames per second. During these experiments, the cameras are operated at 640 x 512 and 10 frames per second. The cameras are mounted with a 50cm baseline.

The INS has an accelerometer, gyroscope, magnetometer, and L1 band GPS receiver. The GPS was not used during any of these experiments. The inertial sensors are all MEMS devices and are relatively low performance. The IMU measurements from this device are available at 50Hz.

7.3 Computing payload

The main computer on board the vehicle has a Core 2 Duo dual-core processor running at 1.86GHz with 4GB of RAM. This computer executes all of the high-level algorithms and interfaces with all of the sensors on board the vehicle. The vision computer described in Section 6 is also mounted on the vehicle.

8 Results

With the control system from Section 4, the state estimation system from Section 5 and the vehicle described in Section 7, a full-closed loop trajectory tracking system is possible.

8.1 Trajectory Tracking

Figure 17 shows the hover performance of the vehicle. This test was conducted by providing the trajectory controller with a path that consists of a single waypoint with a desired velocity of zero. During this 45 second test, the maximum deviation of the vehicle from the setpoint was less than one meter.

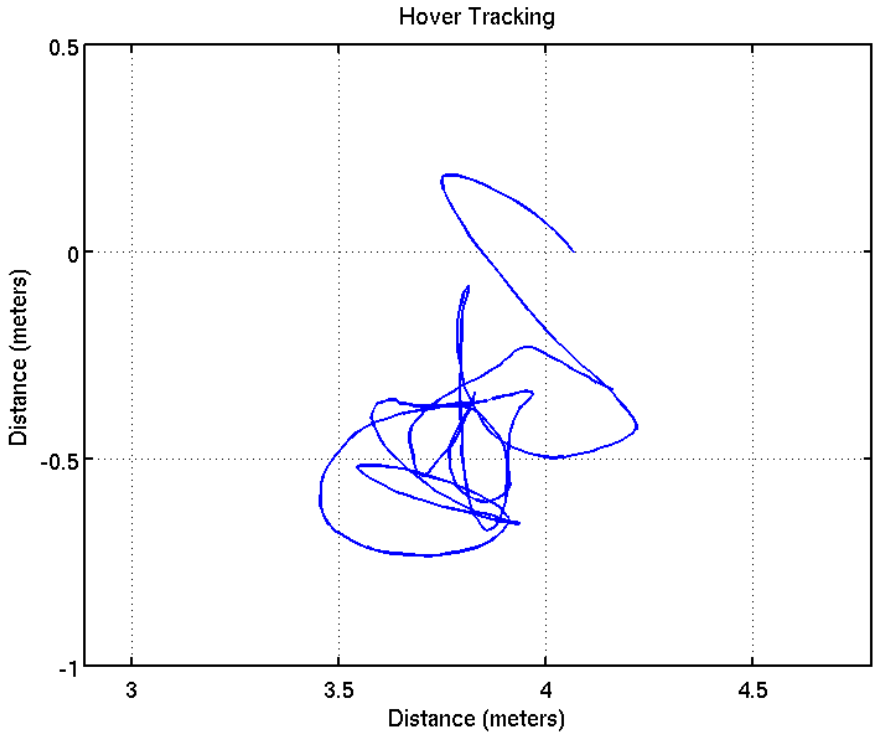

Figure 17: Hover Tracking Performance

Figure 18 shows the vehicle's performance when commanded to track a straight-line trajectory. The desired trajectory is shown in blue, and the vehicles actual trajectory is shown in red with position at the base of the arrow, heading indicated by the direction of the arrow. This trajectory is 10 meters in length and was traversed at 1.5 meters per second. During this test, the cross-track error never exceeded 1 meter.

Figure 19 shows the vehicle's performance when commanded to track a figure-eight trajectory. The notation is the same as in Figure 18. The cross-track error in this test peaked at about 2.0 meters. There is also a noticeable bias in the position of the vehicle. This is, in part, due to the fact that there was a wind blowing in the direction indicated by the arrow.

The total closed-loop performance of a system such as the one described and tested above is affected by the dynamics of all of the constituent systems. The primary limit-

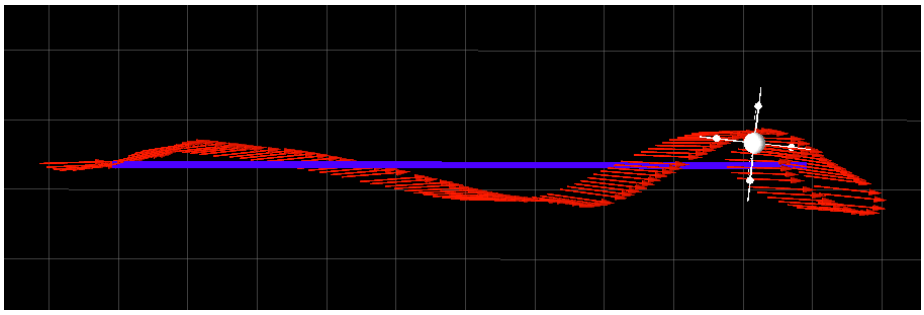

Figure 18: Linear Trajectory Tracking Performance (1 meter grid)

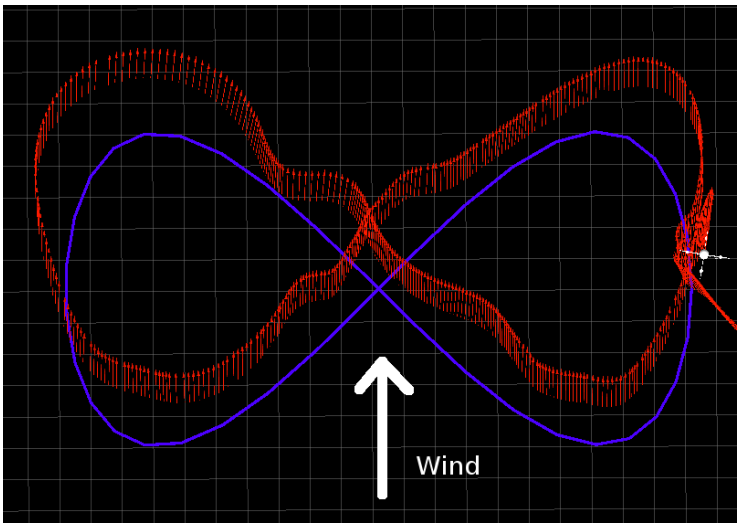

Figure 19: Figure-Eight Trajectory Tracking Performance (1 meter grid)

ing factor to the closed-loop performance of the system described here is the latency of the state-estimation algorithm. This latency is about 130 ms, and the controllers must be tuned to allow for this latency without going unstable. In order to prevent instability, the controllers are tuned to be very conservative. While this is necessary to allow for the state-estimation dynamics, it is detrimental to the closed-loop performance of the system.

Since the state-estimation dynamics are included in the vehicle simulation, very similar effects are observed in the simulation environment. If the state-estimation delay is reduced, the vehicle's simulated performance can be improved significantly by tuning the controllers to be more aggressive.

A secondary effect that limits the total closed-loop performance of the system is the speed of the vehicle's on-board attitude control system. As discussed in Section 4.1, the velocity response is severely limited by the attitude dynamics. The slow response

of the velocity subsystem appears in the trajectory tracking results as a tracking error and the potential for oscillation. By reducing the trajectory control gains, the oscillation of the trajectory tracking is reduced, but the accuracy of the trajectory tracking is simultaneously reduced.

9 Conclusion

The work presented in this report shows a method for controlling a small multi-rotor helicopter based on visual odometry state estimation. This techniques described are demonstrated on a small helicopter with a light payload. A novel hardware solution enabling real-time execution of the algorithms necessary for visual odometry is described and demonstrated.

The control approach presented is very general and can be easily applied to any new vehicle of this type. The controllers are deigned with robustness in mind and are able to handle noise and latency from the state estimation system. The controller can maintain a cross-track error of less than 1 meter while traveling at 1.5 meters per second while tolerating 130ms of state estimation latency. The dynamic modeling technique described requires minimal knowledge of the physical parameters of the system, and instead develops a model based upon experimental data collected from the vehicle in flight. The vehicle model developed responds very similarly to the real vehicle in flight and is therefore very useful for testing algorithms in simulation.

The computing architecture presented allows for the computationally expensive algorithms required for visual odometry to be moved off of the main computer. This architecture enables never before possible computation per gram that enables vision-based state estimation on a class of vehicles that was not previously thought possible.

10 Future Work

While the performance of the system shown in Section 8 is satisfactory, there are several ways that components of the system can be improved in order to increase the performance of the system as a whole.

10.1 State Estimation

As mentioned above, the primary limiting factor in the trajectory tracking performance is the dynamics of the state estimation system. The most detrimental aspect of the stat-estimation system is that there is a significant amount of delay between actual vehicle motion and the motion as reported by the state estimator. If this latency could be removed or reduced, the controllers on the vehicle could be tuned to be more aggressive without the risk of instability. This would result in better velocity tracking and consequently better trajectory tracking.

A method for potentially removing the velocity feedback latency is to introduce a Kalman filter and incorporate the data from the accelerometers on the vehicle into the state estimator. There would be two primary benefits from doing this. The first is that

the frequency of the feedback could be increased significantly. By using the accelerometer to fill in the gaps in the velocity data, the velocity feedback would be available at the frequency at which the accelerometer operates rather than the frequency that the cameras operate at. The second benefit is that the accelerometer data can be used to compensate for the latency of the visual odometry algorithm by extrapolating beyond the most recent visual odometry update. These two benefits would likely improve the performance of the system significantly.

10.2 Control Techniques

If a state-estimator such as the one described in the previous section is implemented, it opens the door for more advanced control techniques to be applied to this system. One possibility is to move from a PID type controller to a linear quadratic regulator (LQR) controller. This type of controller requires the full-state feedback provided by a more advanced state-estimation algorithm, but allows for improved performance of the entire system. It would also be possible to consider non-linear control techniques such as backstepping as described in [14].

A problem with a system that relies on vision-based state feedback is that the state-estimation system has the potential to fail. Of course, it is designed so that this does not happen, but there are times where the accuracy of the state estimate may become poor or perhaps even fail completely. It would be a useful addition to the system if the controllers could be aware of such a failure and compensate appropriately. By doing this, the controllers could be aggressive for maximum performance when a high-quality state estimate is available, but they could also become more conservative to maximize safety and stability when the state estimate is poor or unavailable.

10.3 Vision Processing Optimization

There are several improvements that can be made to the vision processing hardware described in Section 6. There are improvements that can be made to both the hardware and software.

First, there are ways to improve the algorithms that run on board the embedded hardware to take advantage of available hardware. Currently, we utilize the DSP for convolution operations, but it could also be used to calculate the vector distances required for feature matching. In addition to the DSP, the Torpedo COM also has a graphics processing unit (GPU) that could be utilized for some of the highly-parallelizable operations. Specifically, the GPU can be used to perform operations like un-distortion since it is very similar to the graphics operation of texture mapping that the GPU is designed to perform quickly.

This type of software improvement would increase the rate at which image data can be processed by the system. This would enable operation at a higher frame-rate or higher resolution images. Both of these improvements could increase the accuracy and robustness of the visual odometry algorithm.

Second, it is possible to improve the hardware used to implement these algorithms. As described above, the system requires a host computer to read the images from the cameras and facilitate communication between the vision processors. A way that the

system can be improved significantly is by removing the host computer from the system entirely. Since the Torpedo COMs are designed for use in a cell phone, they are capable of interfacing directly with an imaging sensor. By connecting the imaging sensors directly into the Torpedos and introducing a a fifth Torpedo to perform the core visual odometry algorithm it would be possible to eliminate the host computer entirely.

This improvement would make it possible to create a full visual odometry state-estimation solution in about 60 grams of hardware including the cameras. Making the entire visual-odometry system a single piece of hardware would enable visual state estimation on an entire class of vehicles that were previously unable to carry the hardware necessary for such on-board computation.

References

[1] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an rgb-d camera," in *Int. Symposium on Robotics Research (ISRR)*, (Flagstaff, Arizona, USA), Aug. 2011.

[2] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy, "Stereo vision and laser odometry for autonomous helicopters in gps-denied indoor environments," in *Proceedings of the SPIE Unmanned Systems Technology XI*, vol. 7332, (Orlando, F), 2009.

[3] L. D. Minh and C. Ha, "Modeling and control of quadrotor mav using vision-based measurement," in *Strategic Technology (IFOST), 2010 International Forum on*, pp. 70 –75, oct. 2010.

[4] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A system for autonomous flight using onboard computer vision," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 2992 –2997, may 2011.

[5] A. Wu, E. Johnson, and A. Proctor, "Vision-aided inertial navigation for flight control," in *2005 AIAA Guidance, Navigation, and Control Conference and Exhibit*, pp. 1–13, 2005.

[6] M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart, "Vision based mav navigation in unknown and unstructured environments," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 21 –28, may 2010.

[7] G. Conte and P. Doherty, "An integrated uav navigation system based on aerial image matching," in *Aerospace Conference, 2008 IEEE*, pp. 1 –10, march 2008.

[8] R. Voigt, J. Nikolic, C. Hurzeler, S. Weiss, L. Kneip, and R. Siegwart, "Robust embedded egomotion estimation," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 2694 –2699, sept. 2011.

[9] S. Bouabdallah, "Design and control of an indoor micro quadrotor," in *In Proc. of Int. Conf. on Robotics and Automation*, 2004.

[10] P. Pounds, R. Mahony, P. Hynes, and J. Roberts, "Design of a four-rotor aerial robot," in *Proc. 2002 Australasian Conference on Robotics and Automation*, vol. 27, p. 29, 2002.

[11] L. Ljung, ed., *System identification (2nd ed.): theory for the user*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.

[12] G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, "Quadrotor helicopter trajectory tracking control," *Electrical Engineering*, no. August, pp. 1–14, 2008.

[13] B. Kitt, A. Geiger, and H. Lategahn, "Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme," in *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pp. 486–492, IEEE, 2010.

[14] T. Madani and A. Benallegue, "Backstepping control for a quadrotor helicopter," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 3255 –3260, oct. 2006.