AIAA Paper No. 2008-7448, AIAA Guidance, Navigation, and Control Conference, Honolulu, Hawaii, Aug., 2008.

Vision-Based Target Geolocation and Optimal Surveillance on an Unmanned Aerial Vehicle

James A. Ross*, Brian R. Geiger*, Gregory L. Sinsley†,
Joseph F. Horn‡, Lyle N. Long§, and Albert F. Niessner¶
The Pennsylvania State University, University Park, PA 16802.

A real-time computer vision algorithm for the identification and geolocation of ground targets was developed and implemented on the Penn State University / Applied Research Laboratory Unmanned Aerial Vehicle (PSU/ARL UAV) system. The geolocation data is filtered using a linear Kalman filter, which provides a smoothed estimate of target location and target velocity. The vision processing routine and estimator are coupled with an onboard path planning algorithm that optimizes the vehicle trajectory to maximize surveillance coverage of the targets.

The vision processing and estimation routines were flight tested onboard a UAV system with a human pilot-in-the-loop. It was found that GPS latency had a significant effect on the geolocation error, and performance was significantly improved when using latency compensation. The combined target geolocation and path planning system was tested on the ground using a hardware-in-the-loop simulation, and resulted successful tracking and observation of a fixed target. Timing results showed that is is feasible to implement total system in real time.

I. Introduction

Unmanned Aerial Vehicles (UAVs) have been used for many military and civilian applications, with some success. In order for UAVs to demonstrate their full potential, however, they need to become more autonomous. This requires fusing data from all UAV sensors, and using that data to guide and control the aircraft. The more data a control program has, the more likely it is to be able to operate with little to no human intervention. Sinsley *et al.*[1,2] discuss methods for making UAVs both more intelligent and more autonomous. Papers which discuss architectures for improving the intelligence and autonomy of ground vehicles include Hanford and Long,[3] Hanford *et al.*[4] and Janrathitikarn and Long.[5]

One important sensor that many vehicles have is a camera. With a camera, a vehicle can obtain a great deal of information about its surroundings, thereby increasing its level of autonomy. Therefore, this paper will focus on the use of vision data in UAV guidance and control.

Other papers on vision-based control include Dobrokhodov *et al.*[6] In their paper, all image processing work is done on the ground. This allows them to use more complicated image processing routines while still keeping the UAV cost low. Unfortunately, it means that communications dropouts between the UAV and the ground computer can have adverse affects on operation. Although it doesn't deal with UAVs, Baumgartner *et al.*[7] discusses sensor-based navigation of a planetary rover. They show how combining a stereo camera and the robot's odometer provides for more accurate navigation than using the odometer alone.

In this paper a single camera is used as a sensor, and an optimal path planner provides guidance commands to an autopilot. Once a target (a pair of 75 cm diameter red balls) is found using color filtering, its location is determined by a georeferencing process, and its velocity is estimated using a linear Kalman filter. The optimal path planner then determines the path to fly that will keep the target in the field of view of the camera for the greatest possible amount of time. The system was designed such that it is entirely self-contained on

*Graduate Research Assistant, Aerospace Engineering, AIAA Student Member.
†Graduate Research Assistant, Electrical Engineering, AIAA Student Member.
‡Associate Professor, Aerospace Engineering, AIAA Associate Fellow.
§Distinguished Professor, Aerospace Engineering, AIAA Fellow.
¶Senior Research Associate, Emeritus, Applied Research Laboratory.

a small low-cost UAV. All of the image processing, geolocation, and path planning algorithms are calculated by onboard processors. The system does not require any processing on ground-based computers.

The remainder of the paper is organized as follows. Section II discusses algorithms for image processing, Section III deals with finding the world location of the target, Section IV discusses the linear Kalman filter, and Section V provides a overview of the optimal path planner. Section VI provides details on tests of the integrated system including a discussion of the UAV testbed, and results of hardware-in-the-loop and flight tests.

II. Image Processing

At this time, object recognition has not been a major focus of research. Rather, the focus has been on finding a simple target in order to test geolocation, estimation, and path planning algorithms. These algorithms have been designed to be independent of what object recognition algorithm is used, so that more sophisticated object recognition can easily be incorporated in the future. For this reason, a 75 cm red ball was chosen as a target. Red was chosen so that the ball would stand out, and a ball was chosen because it would look the same from all angles. Unfortunately, the flying field used in the tests is lined by red barrels, which look like red balls from certain angles in flight. This made it necessary to use two red balls a set distance apart, so that it would be easier to distinguish the balls from the barrels.

In order to detect the red balls, several algorithms were investigated, including Canny edge detection, Harris corner detection, and color filtering.[8] In the end, the color filtering algorithm was chosen for use, because it performs well, and is simple to implement.

The open source computer vision library, OpenCV,[9] is used for capturing, processing, and saving the aerial images. The library is written in C++ and runs in all 32-bit Windows and POSIX operating systems. It includes many computer vision examples, and the Yahoo! OpenCV community,[10] currently with over 28,000 users, provides additional support.

II.A. Color Filtering

Color filtering is a simple computer vision algorithm for finding colored objects in images. It consists of converting a three-channel RGB image into the Hue-Saturation-Value (HSV) color space. Then, with hue thresholds, H_{min} and H_{max}, and a minimum saturation threshold, S_{min}, each pixel that satisfies both thresholds is marked. When an image is captured from the webcam, it is filtered for hue and saturation resulting in a binary image of regions that meet the filter criteria. This binary image is dilated (using *cvDilate*) to remove small gaps in the filtered image. The dilated image is then searched and all connected areas are given a unique label using *cvFindContours*. Each connected area is called a blob. Each blob is then filtered by size, aspect ratio, solidity, and pixel count so that only blobs with the characteristics of the 75 cm diameter red balls remain. These remaining blobs are used in the target georeferencing algorithm.

II.B. Camera Calibration

Before performing any image processing routines, the camera must be properly calibrated to determine the transformation from pixel coordinates to world coordinates. Fortunately, the Camera Calibration Toolbox for Matlab[11] makes this process fairly straightforward. The first step in the calibration process is to take some pictures of a checkerboard of known size from different angles and distances. Then, using the Matlab graphical user interface provided with the calibration toolbox, the user interactively selects four corners forming a rectangle in the checkerboard pattern. Based on the known physical dimensions of the checkerboard squares, all of the checkerboard corners in the interior of the user-selected areas are extracted from the calibration images and used to determine the unknown intrinsic parameters of the camera. A nonlinear optimization routine finds the set of intrinsic parameters that predicts the corner positions with the least error. Based on the calibration procedure, the focal length, principal point, skew, radial and tangential distortion, pixel error, and camera field of view were determined for the webcam used in this paper. The calibration toolbox provides an inverse function to extract nondimensional world coordinates from pixel coordinates that accounts for radial and tangential distortions. However, it requires an iterated solution. Therefore, only the focal length and principal point are used to recover world coordinates. Significant error is only introduced for the webcam when a target is in the extreme corners of the image.

III. Target Georeferencing

Algorithms for real-time georeferencing onboard the Penn State University / Applied Research Laboratory Unmanned Aerial Vehicle (PSU/ARL UAV) system have been developed for autonomous operation. Using a downward-pointing USB webcam and streaming aircraft state telemetry from an autopilot, a target vector toward a point of interest in the image frame is calculated. Combined with a virtual terrain model of the flight area, an approximate GPS location of a point on the ground is then estimated.

Terrain data is obtained using the USGS Seamless Data Distribution System[12] and then converted into an ArcInfo ASCII Grid for ease of processing. The data is read into memory on the onboard computer and the data points are converted into a local three-dimensional coordinate system. This is used in an algorithm to calculate the geographic location of a ground target as seen from the UAV camera.

The position and orientation of the onboard camera can be estimated by obtaining the position and orientation of the UAVs autopilot system in the telemetry data stream. Then, using a vector from the camera to the target of interest in the image frame, the three-dimensional points $\boldsymbol{l_a}$ (camera position) and $\boldsymbol{l_b}$ (point on target vector) are calculated in the local coordinate system. Using three of the closest grid points $\boldsymbol{p_0}$, $\boldsymbol{p_1}$, and $\boldsymbol{p_2}$ to create a plane and the two points ($\boldsymbol{l_a}$ and $\boldsymbol{l_b}$) on the vector from the UAV to the ground target, a point of intersection (the ground target location) is found with the formula $\boldsymbol{l_a} + (\boldsymbol{l_b} - \boldsymbol{l_a})t$ where t is found by solving:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} x_a - x_b & x_1 - x_0 & x_2 - x_0 \\ y_a - y_b & y_1 - y_0 & y_2 - y_0 \\ z_a - z_b & z_1 - z_0 & z_2 - z_0 \end{bmatrix}^{-1} \begin{bmatrix} x_a - x_0 \\ y_a - y_0 \\ z_a - z_0 \end{bmatrix} \tag{1}$$

All values in this equation are components of the five three-dimensional points stated above. The algorithm then iterates on planes in the local grid, using the last calculated point of intersection to update the three grid points in the next plane, to find the exact ground location of the target. The plane directly below the UAV is used for the first guess and convergence usually occurs within 1 or 2 iterations. The local coordinates are then converted into GPS coordinates. Figure 1 provides a graphical view image frame projected onto the terrain model.

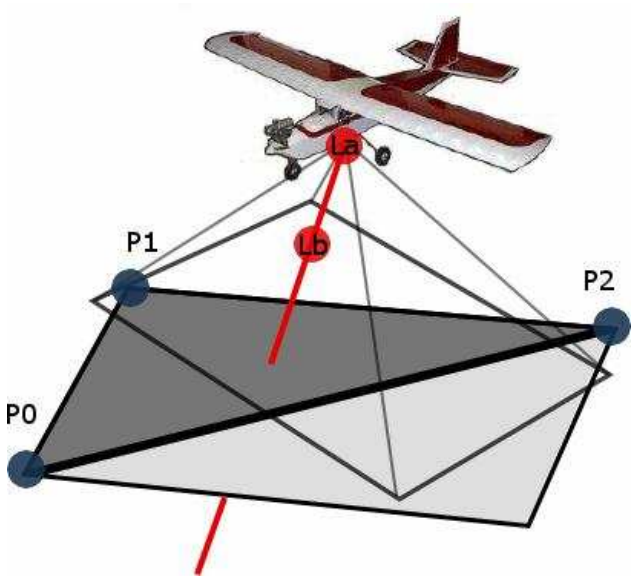


Figure 1. Graphical depiction of camera view on terrain

By combining the vision and georeferencing software, we have demonstrated the ability to autonomously identify and locate a target consisting of two 75 cm diameter red balls in flight in real time. Figure 2 shows an image taken by the UAV with a bounding box around both balls. A red barrel at the bottom of the image is excluded because it is too large and has an aspect ratio greater than the threshold. By looking for two red blobs separated by a certain distance and of similar sizes, the number of false positives returned by the target recognition algorithm is greatly reduced. With this target configuration, false positives are rare but

do occur occasionally. They can be filtered out by ignoring target locations that are a large distance away from where they have previously been observed. Figure 3 shows a ground track taken from the same run with a red quadrilateral over the camera's field of view. (Note that the background image is a USGS image of our flying field).

Figure 2. Aerial photo of a correctly identified dual red ball target

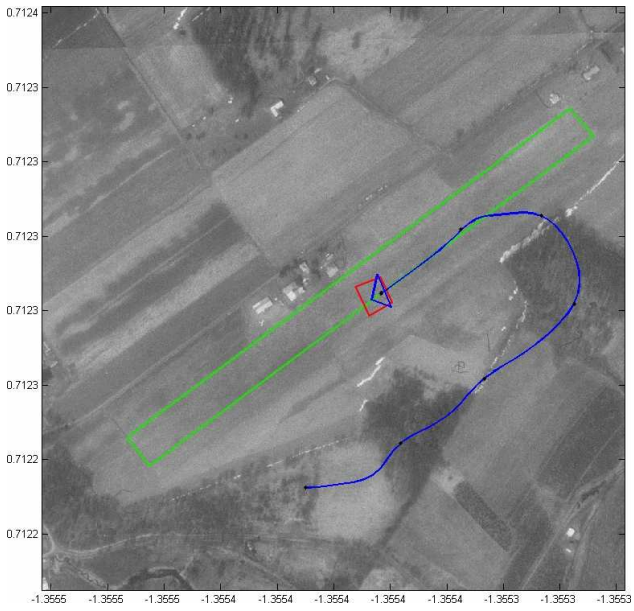


Figure 3. Ground track of UAV with the camera's field of view

IV. Kinematic Estimation

Once a target is located, a linear discrete time Kalman filter is used to smooth position estimates and estimate the velocity. In order to use a linear Kalman filter, target coordinates are transformed from latitude and longitude to a local coordinate system. The coordinate system has as its origin a point near the target's initial location. The target's position is then measured in meters north and east of the initial estimate. The

state of the system is then:

$$x = \begin{bmatrix} n \\ e \\ v_n \\ v_e \end{bmatrix} \tag{2}$$

The target is modeled as a second order Newtonian system by the equation:

$$x_{k+1} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} u \tag{3}$$

where Δt is the time step between updates, which is 0.25 seconds for real-time implementation, and u is the acceleration input, which is modeled as Gaussian white noise:

$$u \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_{a_n}^2 & 0 \\ 0 & \sigma_{a_e}^2 \end{bmatrix} \right) \tag{4}$$

In this implementation the acceleration standard deviations are set to $\sigma_{a_n} = \sigma_{a_e} = 0.1\ m/s^2$, which yields a slowly maneuvering target. The output equation is:

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} v \tag{5}$$

where y is the output of the image processing program (converted to meters from the reference point), and v is the measurement error, which is modeled as Gaussian white noise:

$$v \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_n^2 & 0 \\ 0 & \sigma_e^2 \end{bmatrix} \right) \tag{6}$$

The output noise standard deviations are set to $\sigma_n = \sigma_e = 5\ m$.

The initial guess of the target's state is usually set to the reference location, with zero velocity. The covariance of the error in the initial state is set to:

$$P = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix} \tag{7}$$

which corresponds to a standard deviation of 5 m in position and 0.1 m/s in velocity, and no correlation between the states. Given this information, the target's location and velocity at each time step (and the covariance of the error estimate) are then found using the discrete time Kalman filter equations found in many textbooks, including Simon.[13]

V. Path Planning

The path planning algorithm is based on a direct method called Direct Collocation with Nonlinear Programming (DCNLP), first introduced by Dickmanns.[14] The work presented here is based on a paper by Hargraves and Paris.[15] A complete description of the method and implementation on the PSU/ARL UAV is given in Geiger *et al.*[16] A brief description is given here.

A receding horizon control approach is taken with DCNLP providing the trajectory over the horizon time T_h. The horizon is discretized into n equal segments of τ seconds. The endpoints of these segments are the nodes. The states and controls from the equations of motion are defined at these nodes, and the segments between the nodes are approximated. Cubic Hermite polynomials approximate the state

trajectories, and controls are represented linearly. Hermite polynomial segments are used for the state interpolation because a segment can be defined completely using the endpoint values and first derivatives at the endpoints. This corresponds to the state vector $\boldsymbol{x}$ and its derivative $\dot{\boldsymbol{x}}$ which are given at each node. To ensure the interpolating polynomials accurately represent the state equations between the nodes, a constraint is applied at each segment midpoint such that the derivative of the interpolating polynomials must match that of the state equations. Additional constraints are applied to the states and controls at each node so that the resulting trajectory does not exceed vehicle capabilities such as maximum turn rate, airspeed, or bank angle. The SNOPT[17] nonlinear solver then minimizes a scalar objective function based on the states and controls at the nodes to compute the final path.

The state model used here is a simple two dimensional planar motion model that has been found to work well in practice.[16] The states are are North and East position of the UAV and target (n, e, n_t, e_t), airspeed of the UAV (V_t), and heading of the UAV (ψ). The controls are longitudinal acceleration command (u_l) and bank angle command (u_ϕ). The aircraft equations account for a constant wind speed (V_{wind_N}, V_{wind_E}).

$$\begin{aligned}
\dot{x} = \dot{\boldsymbol{x}}_{u_1} &= V_t \cos(\psi) - V_{wind_N} \\
\dot{y} = \dot{\boldsymbol{x}}_{u_2} &= V_t \sin(\psi) - V_{wind_E} \\
\dot{V}_t = \dot{\boldsymbol{x}}_{u_3} &= u_l \\
\dot{\psi} = \dot{\boldsymbol{x}}_{u_4} &= g \tan(u_\phi)/V_t \\
\dot{x}_t = \dot{\boldsymbol{x}}_{t_1} &= V_{tgt_N} \\
\dot{y}_t = \dot{\boldsymbol{x}}_{t_2} &= V_{tgt_E}
\end{aligned} \tag{8}$$

The objective function used to maximize target observation time is given in Eq. 9.

$$J = \int_{t_o}^{t_f} \left[w_1 u_l^2 + w_2 u_\phi^2 + w_3((x - x_t)^2 + (y - y_t)^2) + w_4 J_{tiv} \right] \tag{9}$$

The first two terms penalize control effort (longitudinal acceleration and bank angle), and the third term weights the square of the distance to the target. The fourth term, J_{tiv}, is a "target-in-view" cost function which has its minimum value when the target is at the center of the image plane of the onboard camera and reaches its maximum value when the target is out of the camera frame. This causes the optimization to favor pointing the camera so that the target is directly in the center of the field of view.[16]

V.A. Path planning and target geolocation integration

To allow both the path planner and target localizer to run simultaneously, several issues must be addressed. Both programs require UAV telemetry data, and both need to communicate with each other. The path planner needs to send commands back to the autopilot as well. Figure 4 shows the basic layout of the required connections. Timely communication between the processes is paramount, so a shared memory scheme was chosen. The host controller receives telemetry from the autopilot over a serial connection and places the data structure in a shared memory location so the path planner and target geolocation processes can access it. By continually updating the telemetry, each process always has the most recent available data without the need for polling or requesting it. The target location shared memory is continually updated with estimated target location from the Kalman filter. The path planner reads the target location information and creates a path. When path generation is finished, the path planner loads the path into shared memory, and then sets the shared update flag. The host controller checks this flag each cycle; if set, the shared memory containing the path is read and sent to the autopilot. To avoid deadlock issues, no process depends on another to continue running. Timing data is recorded for all processes to measure how the integrated setup performs. These are presented in the Results section.

VI. Testing

The integrated vision based guidance system has been tested using the UAV platform at the Penn State UAV Lab.[2,18] The vision and estimation systems have been tested using hardware-in-the-loop simulations, and have been run in real time onboard the aircraft with a human pilot in the loop. The complete system, including the path planning program, has been tested in hardware-in-the-loop.

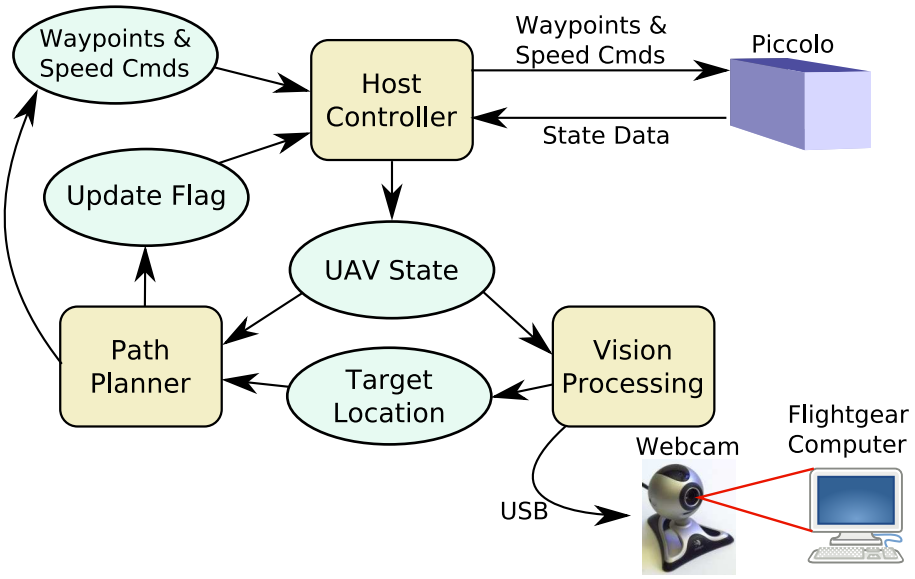


Figure 4. Schematic of the hardware-in-the-loop simulation

VI.A. Platform

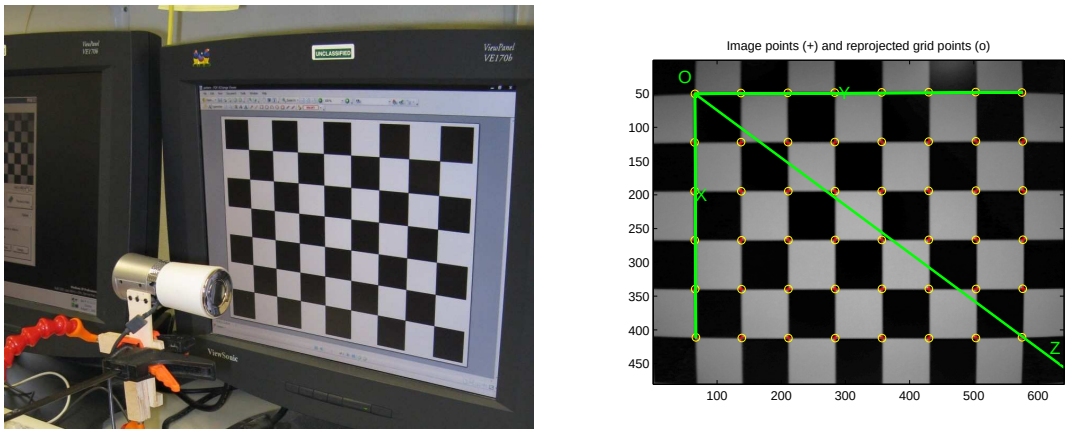
The Penn State UAV Lab's aerial platform consists of heavily modified SIG Kadet Senior model aircraft (Figure 5), Cloud Cap Technology Piccolo Plus Autopilot, Ampro ReadyBoard single board computer, a ground station laptop, and a Logitech Quickcam webcam. In the airborne configuration, the Piccolo is connected to the UAV's servos, and is controlled by a human operator on the ground using a pilot console supplied with Cloud Cap Technology's base station. All image processing is done in real time using the onboard webcam and the onboard processor. More details on the UAV platform can be found in Miller *et al.*[18] and in Sinsley *et al.*[2]

Figure 5. SIG Kadet UAVs

VI.B. Hardware-in-the-loop simulation description

In the hardware-in-the-loop (HIL) setup, the autopilot is connected to a computer that simulates flight dynamics. The Ampro ReadyBoard is connected to the autopilot and runs the image processing and path planning software. The camera is connected to the USB port of the ReadyBoard and is pointed at a LCD which displays aerial images generated by Flightgear,[19] thus simulating what it would actually see in flight. Figure 6(a) shows the setup. Figure 6(b) shows an example of alignment calculation using the camera

calibration toolbox.



(a) Webcam with on-screen calibration pattern

(b) Camera calibration result

Figure 6. Webcam HIL rig and calibration result

Several problems present themselves with this setup. The most basic is the choice of screen. A CRT screen cannot be used as the webcam will pick up flickering as the screen is redrawn. While tuning a camera to prevent flickering is possible, most cheap webcams do not have this option. Therefore, an LCD is used to avoid flickering. The second problem to solve is correct positioning of the camera; it should be aligned with the normal vector of the LCD screen's surface. To compute the orientation of the camera with respect to the screen's surface, the camera calibration toolbox is used with the calibration pattern (shown in Figure 6(a)). The required roll, pitch, and yaw corrections for the camera of the camera is easily computed. An alignment to within 0.3 degrees in all axes was attained. The field of view in Flightgear is then set to match the field of view of the camera.

The final component of the HIL system is the actual visualization of the terrain. Flightgear comes with default world scenery that is fairly sparse. It does have limited support for photorealistic scenery over small areas. Using 1 ft/pixel resolution aerial photos of the flying field from Pennsylvania Spatial Data Access (PASDA),[20] a 4096 ft^2 area centered at the runway was imported into Flightgear using the *photomodel* command (available when Flightgear is compiled from the source). Note that the area should be a power of two (in this case, 4096 pixels square) for the texture to be properly displayed in Flightgear. Shown in Figure 7 is a comparison between actual picture recorded in flight and visuals produced by Flightgear using the photorealistic scenery. The webcam reduces the contrast and detail of the Flightgear visuals, however it is still usable by the target recognition algorithm. The darkening in the top half of Figure 7(c) is due to the viewing angle limitations of the LCD screen. One problem with using photorealistic scenery in Flightgear is that it is difficult to apply the aerial photo over an elevation model. Therefore, the model used here is simply a flat plane. The focus of the HIL simulation is not to test absolute localization accuracy, but to provide a platform for debugging the system in a "realistic-enough" environment.

(a) Actual image captured in flight

(b) Flightgear screen shot

(c) Flightgear as seen through the webcam

Figure 7. Comparison of actual and simulated aerial views

VI.C. Vision and Estimation

The initial test using the HIL simulation was a simple target observation by a UAV flying a rectangular pattern over the target. The target was approached from two different directions. Figure 8 shows the geolocation results for three passes in each direction. A small red line pointing from the center of each UAV location indicates yaw angle. Figure 9(a) shows the covariance over time, and Figure 9(b) shows the absolute error over time.

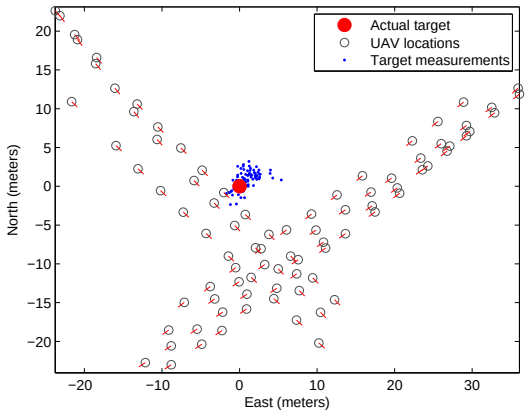
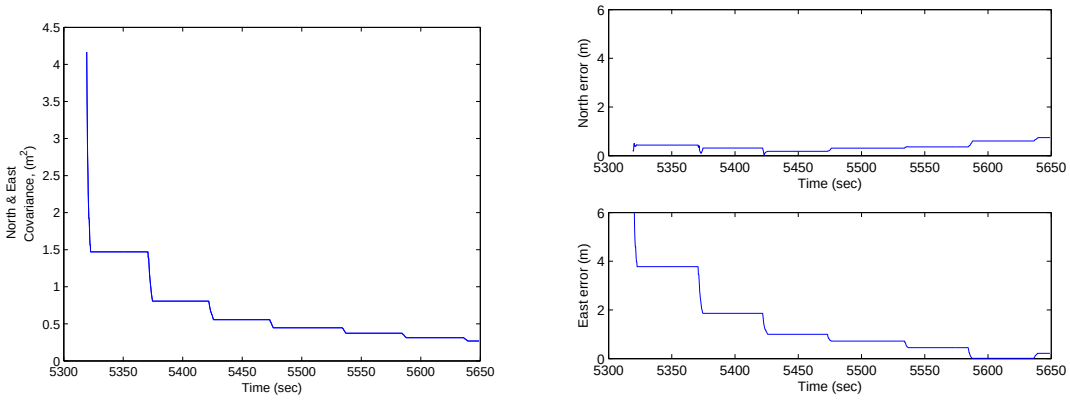


Figure 8. HIL simulation geolocation results



(a) Covariance over time

(b) Absolute error over time

Figure 9. Accuracy of the target geolocation

The HIL simulation is useful for system debugging, however the geolocation results obtained are a little too clean. A manually piloted flight test was conducted to test the accuracy of the target geolocation algorithm. The UAV was flown over the target several times from multiple angles to capture observations of the target. Figure 11(a) shows the target observations. They are fairly scattered, seemingly clustering *around* the target, especially when compared to the HIL simulation results. This suggests an unknown bias in the measurements which might be caused by latency that is not simulated in the HIL setup. For example, if the GPS measurement is delayed, then the target will always appear to be behind its actual location in the direction of the UAV's flight path over the target.

A test was conducted on the autopilot GPS to determine its latency. The method described by Bouvet and Garcia[21] uses a construction compactor vehicle with wheel encoders on each side. By accelerating and decelerating the compactor and comparing data from the wheel encoders and onboard GPS, GPS latency is calculated. A similar setup for the autopilot is beyond the means of the PSU UAV lab, therefore the Network Time Protocol (NTP)[22] is used. NTP is a protocol designed to synchronize computers over the Internet (or any network). Timing accuracies on the order of several milliseconds or less can be achieved depending on the network. NTP compares the times of several servers nearby on the network and compute the actual time accounting for network lag. By adding the autopilot GPS as a time source, NTP is able to determine latency

compared with the other time servers. There are many assumptions made in this test, two of which are that time latency corresponds with position latency and latency of the serial link between the computer and the autopilot is neglected. However, as shown below, assuming GPS latency exists improves the accuracy of the results. The latency measured with this setup is around 430 milliseconds. Figure 10 shows a histogram of the results.

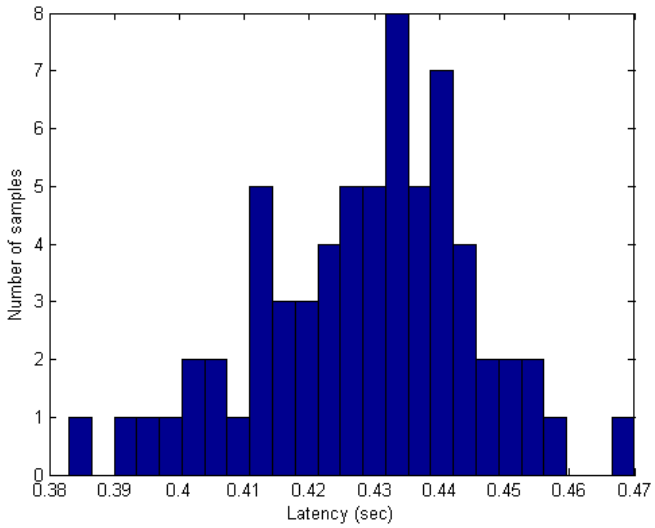
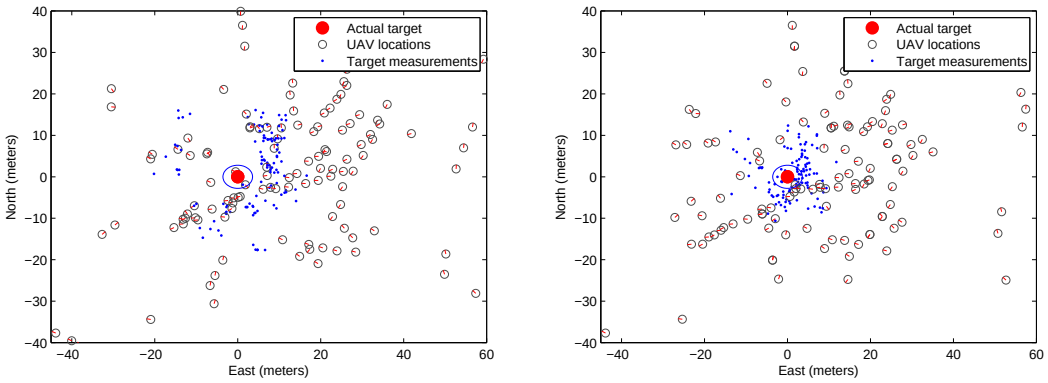


Figure 10. Measured GPS Latency

Post-processing the results by using the GPS position estimate 430 ms into the future results in the following target observations shown in Figure 11(b). Note that the points are more tightly clustered around the actual target location. Finally, the absolute position error is approximately reduced by a factor of approximately two, as seen in Figure 12.



(a) Raw target observations

(b) Target observations using latency compensated GPS measurements

Figure 11. Manual flight target observations

Estimates of target velocity from the Kalman filter taken in flight were very poor due to bad choices of process noise and measurement noise. After this problem was discovered, the data was post-processed using the values found in Section IV. Figure 13 shows error in the Kalman filter's estimate of position. Note that the position error grows when the target is not in the UAV's field of view, then returns to a very small value when the target once again enters the field of view. This is because it is assumed that the target moves with a constant speed when it is not in the UAV's field of view. So, if the speed estimate is incorrect, the estimate of the target's position will diverge. Figure 14 shows estimates of the target's speed for the same run. The long periods of time where speed is constant correspond to when the target is not in the UAV's field of view. The target is not moving in this flight, so the low speeds (less than 1 m/s), which are observed

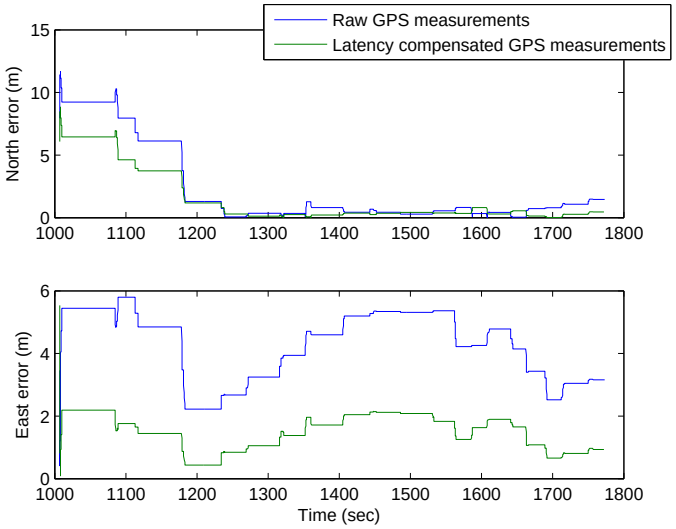


Figure 12. Comparison of absolute error between raw and latency compensated GPS measurements

are to be expected. Finally, Figure 15 shows the covariance of position and velocity estimates over time. The covariance of the velocity estimate grows when the target is not in view, then shrinks when the target comes into view. This makes sense, since the Kalman filter will not have much confidence in its estimate if it is not receiving new data. Figure 16 shows a zoomed in view of this plot, along with a plot of velocity. The velocity is constant when the target is not in view, so it is possible to tell exactly where the target enters and leaves the field of view.

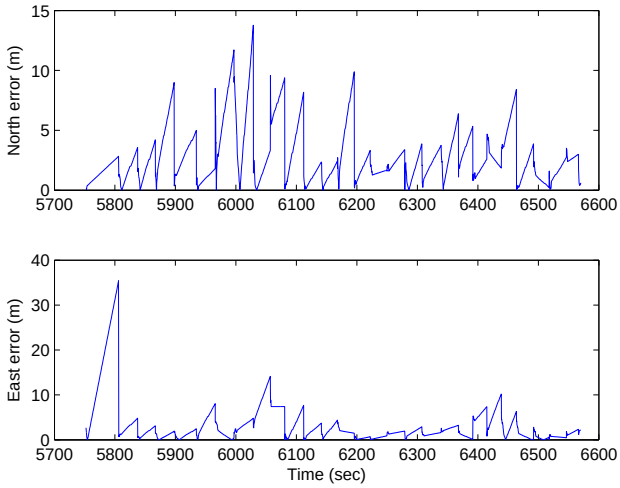


Figure 13. Error in the Kalman filter's estimate of position

Latency in GPS measurements is not the only source of error. The PSU/ARL UAV is not equipped with a magnetometer so there is no direct measurement of the aircraft yaw angle. The autopilot Kalman filter can estimate the yaw angle using the combination of GPS, inertial, and airspeed measurements. However, it may take some time for the system to accurately account for changing winds which cause the aircraft to sideslip. In strong fluctuating winds there can be significant errors in the estimated yaw angle. Inaccuracies in yaw angle can cause the target to appear to move perpendicularly to the flight path of the UAV as the UAV flies overhead. Errors in altitude can cause the target to appear to move from ahead of to behind the UAV as it overflies the target. Here, a laser altimeter would help for low altitude missions. Misalignments in camera installation also can cause errors, however these are likely to be small and can be calibrated for if required.

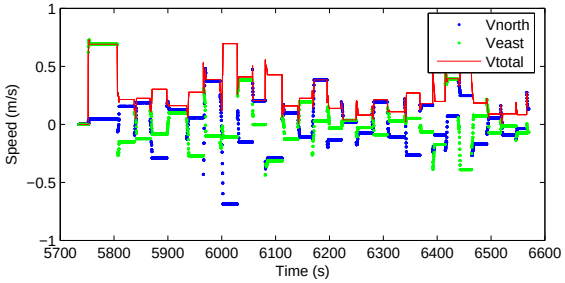


Figure 14. Kalman filter estimate of velocity

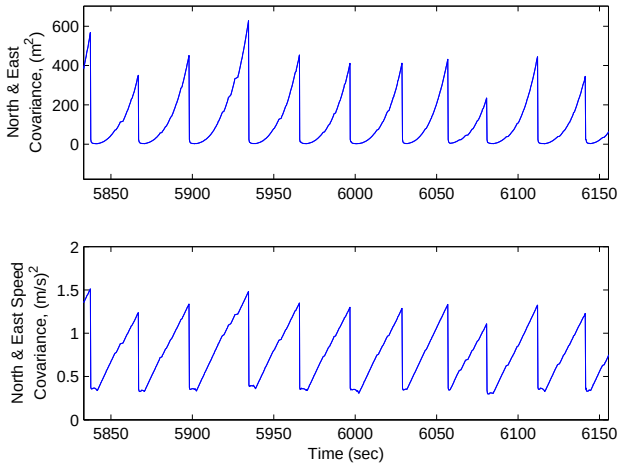


Figure 15. Covariance of the Kalman filter's estimate of position and velocity

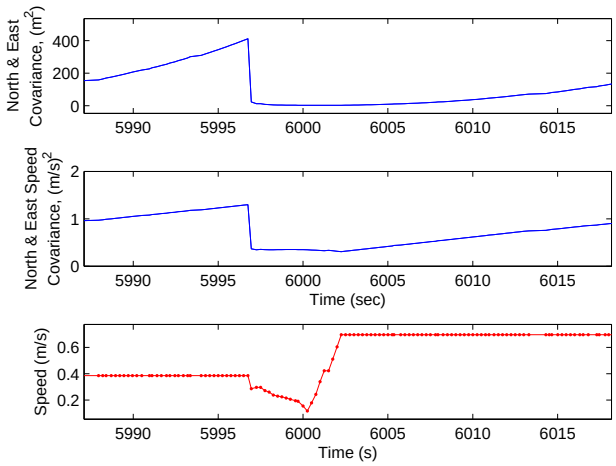
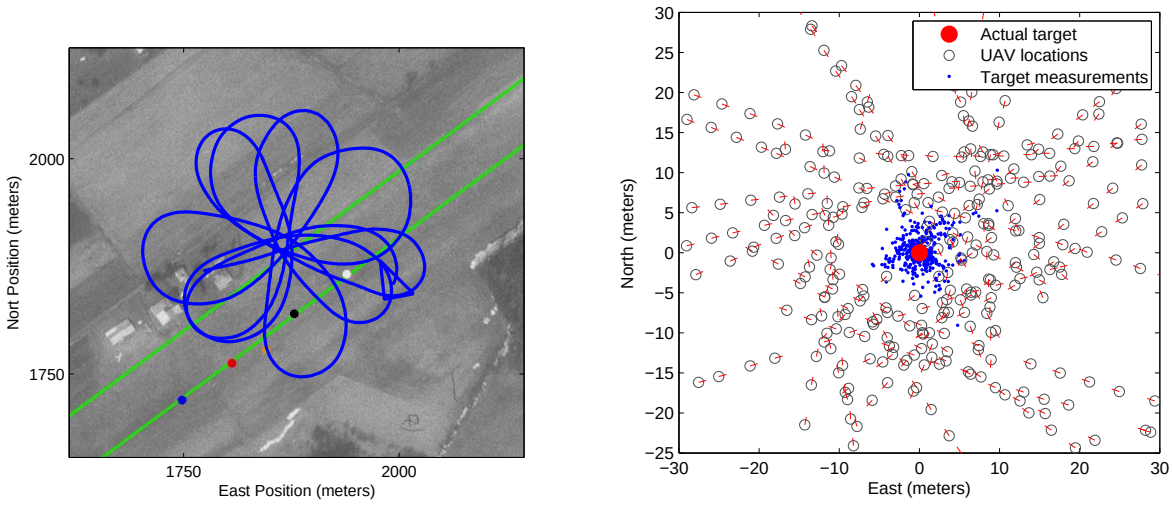


Figure 16. Covariance of the Kalman filter's estimate of position and velocity(zoomed-in view)

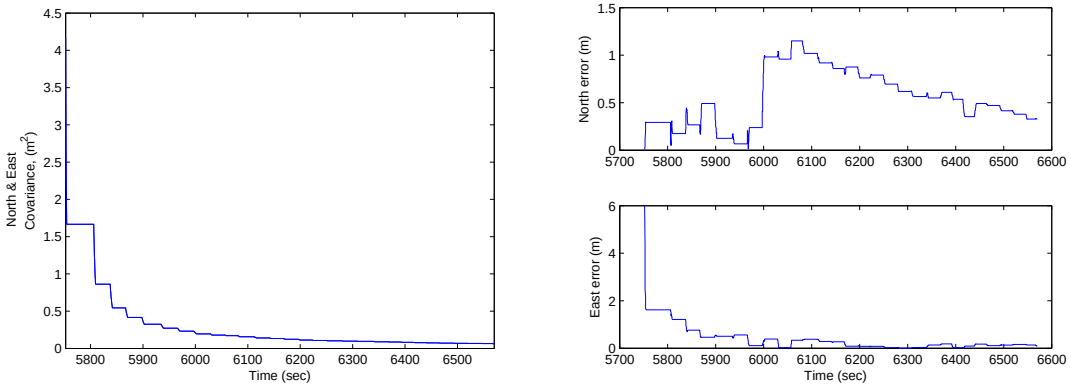
VI.D. Path Planning

The integrated path planner and target geolocation code results are presented here for the HIL simulation test. Figure 17(a) shows the ground path of the UAV; the path planner generates a clover leaf pattern similar to paths observed in previous studies where the target location was given.[16] Figure 17(b) shows individual target observations and the positions of the UAV when the observations were made. Figure 18 shows covariance and absolute error over time. Note that the covariance is smaller than the absolute error. This means that the filter is overconfident in its estimates. This is probably due to a poor initial guess of the covariance or due to poor modeling of the process noise and/or the measurement noise.



(a) Ground track (b) Target observations

Figure 17. Integrated path planner and target geolocation in HIL simulation

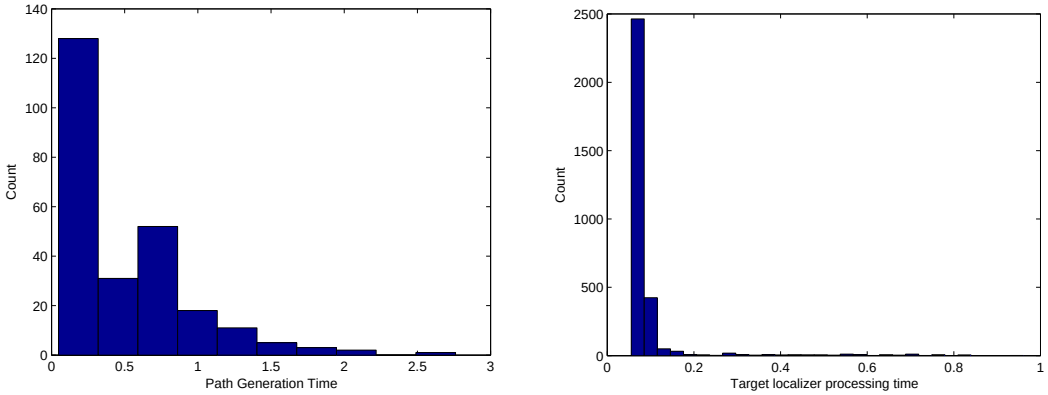


(a) Covariance over time (b) Absolute error over time

Figure 18. Accuracy of the target geolocation

The path planner and target localizer work well together in the HIL simulation. This particular test case ran the target localizer at 4 Hz and the path planner using 7 nodes, a 20 second horizon length, and a 2 second horizon update time. This particular configuration for the path planner was shown to work well in previous work.[16] Figure 19 shows timing results for both the path planner and target localizer. Note that in only 3 out of 251 instances does the path generation time exceed the horizon update interval of 2 seconds. Additionally, these excursions are within 0.5 seconds of the update interval and have a negligible effect on the real-time performance of the path planner. The target geolocation algorithm showed an average processing time of 0.086 seconds per run but with several runs taking as long as 1 second (note it is expected to run at 4 Hz). These extended processing times are most likely due to the path planner running and consuming all available processing resources. However, the accuracy of the target geolocation is not affected because

the UAV state data is saved at the beginning of the run and thus does not change throughout (even as the shared memory is updated). The Kalman filter can be adversely affected by this if it is trying to estimate the target velocity. These timing results show that both the path planner and the target geolocation algorithm are able to run simultaneously at the expected update rates.



(a) Path planner processing time (b) Target geolocation processing time

Figure 19. Histogram of processing times

VII. Conclusions

This paper has presented an integrated vision-based path planning system for UAVs. The system can detect 75 cm diameter red balls and estimate their position and velocity. Based on this data, the path planner determines and optimal path to fly to keep the targets in the camera's field of view for a maximal amount of time. The vision processing and estimation routine has been tested in the air with a human pilot in control of the aircraft. Errors in the location of the target on the order of 4 meters have been obtained when GPS lag is not accounted for. Accounting for the lag in GPS measurements reduces this error further to the order of 2 meters. For safety reasons, the complete integrated system has been tested on the ground using a hardware-in-the-loop simulation. Using data from the vision processing system, the path planning software was able to generate a cloverleaf pattern that would keep the target in the camera's field of view for as much time as possible.

Future work will include adding the ability to track multiple targets and the ability to identify targets of different classes (i.e. being able to tell the difference between a red ball and a red truck). Also, fusing data from multiple sensors, such as a camera and a laser range sensor will be investigated. Using a laser range sensor as opposed to a terrain map, should yield a better estimate of target location, since it will be possible to measure very precisely how far the UAV is from the target. Collaborative work between multiple UAVs and between UAVs and ground vehicles is also under investigation. With multiple vehicles more targets can be tracked, and targets can remain in view of at least one vehicle for as much time as possible.

Acknowledgment

This work was partially supported by the Intelligent Systems Lab at the Penn State Applied Research Laboratory (ARL/PSU). It was also supported by the ARL/PSU Exploratory and Foundational (E&F) Research Program.

The authors would like to thank Nathaniel Rice for his assistance with the flight test portion of this paper.

References

[1]Sinsley, G. L., Miller, J. A., Long, L. N., Geiger, B. R., Niessner, A. F., and Horn, J. F., "An Intelligent Controller for Collaborative Unmanned Air Vehicles," *IEEE Symposium Series in Computational Intelligence*, Honolulu, Hawaii, April 2007.

[2]Sinsley, G. L., Long, L. N., Niessner, A. F., and Horn, J. F., "Intelligent Systems Software for Unmanned Air Vehicles," *46th AIAA Aerospace Sciences Meeting*, No. AIAA-2008-0871, Reno, NV, Jan 2008.

[3]Hanford, S. D. and Long, L. N., "Evaluating Cognitive Architectures for Unmanned Autonomous Vehicles," *22nd Conference on AI, Association for the Advancement of Artificial Intelligence*, Vancouver, Canada, July 2007.

[4]Hanford, S. D., Janrathitikarn, O., and Long, L. N., "Control of a Six-Legged Mobile Robot Using the Soar Cognitive Architecture," *46th AIAA Aerospace Sciences Meeting*, No. AIAA-2008-0878, Reno, NV, Jan. 7-10 2008.

[5]Janrathitikarn, O. and Long, L. N., "Gait Control of a Six-Legged Robot on Unlevel Terrain using a Cognitive Architecture," *IEEE Aerospace Conference*, Big Sky, Montana, Mar. 1-8 2008.

[6]Dobrokhodov, V., Kaminer, I., Jones, K., and Baer, W., "Target Tracking and Motion Estimation Using Imagery Provided by Low Cost UAVs," *Unmanned Systems North America*, 2006.

[7]Baumgartner, E. T., Leger, F. C., Schenker, P. S., , and Huntsberger, T. L., "Sensor-Fused Navigation and Manipulation From a Planetary Rover," *SPIE Conference on Sensor Fusion and Decentralized Control in Robotic Systems*, Boston, Massachusetts, November 1998.

[8]Trucco, E. and Verri, A., *Introductory Techniques for 3-D Computer Vision*, Prentice Hall, Upper Saddle River, NJ, 1998.

[9]"OpenCV 1.0," Available at: http://www.intel.com/technology/computing/opencv, Last accessed on August 3, 2008.

[10]"Yahoo! OpenCV Community," Available at: http://tech.groups.yahoo.com/group/OpenCV, Last accessed on August 3, 2008.

[11]"Camera Calibration Toolbox for Matlab," Available at: http://www.vision.caltech.edu/bouguetj/calib_doc, Last accessed on August 4, 2008.

[12]"USGS Seamless GIS Data," http://seamless.usgs.gov, Last accessed on August 3, 2008.

[13]Simon, D., *Optimal State Estimation*, John Wiley and Sons, 2006.

[14]Dickmanns, E. D. and Well, K. H., "Approximate Solution of Optimal Control Problems Using Third Order Hermite Polynomial Functions," *Proceedings of the IFIP Technical Conference*, Springer-Verlag, London, UK, 1974, pp. 158–166.

[15]Hargraves, C. R. and Paris, S. W., "Direct Trajectory Optimization Using Nonlinear Programming and Collocation," Vol. 10, No. 4, July/August 1987, pp. 338–342.

[16]Geiger, B. R., Horn, J. F., Sinsley, G. L., Ross, J. A., and Long, L. N., "Flight Testing a Real Time Implementation of a UAV Path Planner Using Direct Collocation," *AIAA Guidance, Navigation, and Control Conference*, No. AIAA-2007-6654, Hilton Head, SC, 2007.

[17]Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Journal on Optimization*, Vol. 12, No. 4, 2002, pp. 979–1006.

[18]Miller, J. A., Minear, P. D., Niessner, A. F., DeLullo, A. M., Geiger, B. R., Long, L. N., and Horn, J. F., "Intelligent Unmanned Air Vehicle Flight Systems," *Journal of Aerospace Computing, Information, and Communication (JACIC)*, Vol. 4, No. 1, May 2007.

[19]"Flightgear Flight Simulator," Available at: http://www.flightgear.org, Last accessed on August 3, 2008.

[20]"Pennsylvania Spatial Data Access," Available at: http://www.pasda.psu.edu/, 2008, Last accessed: August 3, 2008.

[21]Bouvet, D. and Garcia, G., "Gps Latency Identification by Kalman Filtering," *Robotica*, Vol. 18, No. 5, 2000, pp. 475 – 485.

[22]"Network Time Synchronization Research Project," Available at: http://www.cis.udel.edu/ mills/ntp.html, Last accessed on August 3, 2008.