JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 6, October 2009

Real-Time Vision Sensor for an Autonomous Hovering Micro Unmanned Aerial Vehicle

Beau J. Tippetts* , Dah-Jye Lee† , Spencer G. Fowers*, and James K. Archibald†
Brigham Young University, Provo, Utah 84602

DOI: 10.2514/1.40185

Vision algorithms were implemented on an field programmable gate array to provide additional information to supplement the insufficient data of a standard inertial measurement unit in order to create a previously unrealized completely onboard vision system for micro-unmanned aerial vehicles. The onboard vision system is composed of an field programmable gate array board, and a custom interface daughterboard which allow it to provide data regarding drifting movements of the micro-unmanned aerial vehicle not detected by inertial measurement units. The algorithms implemented for the vision system include a Harris feature detector, template matching feature correlator, similarity-constrained homography by random sample consensus, color segmentation, radial distortion correction, and an extended Kalman filter with a standard-deviation outlier rejection technique. This vision system was designed specifically for use as an onboard vision solution for determining movement of micro-unmanned aerial vehicles that have severe size, weight, and power limitations. Results show that the vision system is capable of real-time onboard image processing with sufficient accuracy to allow a micro-unmanned aerial vehicle to control itself without power or data tethers to a groundstation.

Nomenclature

G	autocorrelation matrix
H	image height, pixels
I	grayscale image intensity value
K	Kalman gain matrix
P	estimate covariance matrix
r	radius, pixels
S	innovation covariance matrix
T	state vector
U	innovation vector
W	image width, pixels
x	image column pixel index in image
y	image row pixel index in image
Z	sensor input vector

Received 27 August 2008; accepted for publication 12 August 2009.

* PhD Candidate, Electrical and Computer Engineering, Brigham Young University, Provo, UT 84602, USA.
† Professor, Electrical and Computer Engineering, Brigham Young University, Provo, UT 84602, USA.

Greek

α	distortion correcting polynomial coefficients
Θ	angle of rotation, deg
δ	displacement, pixels
∇	image intensity gradient

Superscripts

$'$	variable pertains to subsequent image
$\hat{}$	estimate
a	color variable

Subscripts

c	center of mass
d	distorted
f	feature variable
h	horizontal component
j	time index variable, frames
u	undistorted
v	vertical component

I. Introduction

REAL-TIME vision sensors are a key component for many unmanned vehicles [1–4]. Real-time vision sensors allow unmanned systems to passively interact with their environment in a way that other sensors cannot. Systems like those described by Kjaer-Nielsen et al. [1], Silva et al. [2], Martins et al. [3], and Hirai et al. [4] implement real-time vision systems to help control unmanned vehicles that do not have severe weight and power restrictions as do micro unmanned aerial vehicles (micro-UAVs). A real-time vision sensor that meets the weight and power constraints of micro-UAVs dramatically increases their functionality. Ettinger et al. [5] suggest that systems based on the use of image sensors and processors are the only practical solution to solving aerodynamic obstacles to flight for micro-UAVs. Barrows [6] suggests that proper vision sensors would allow micro-UAVs to maneuver through complex urban environments. With this ability, micro-UAVs become more practical solutions for many suggested large UAV applications, such as crop dusting, remote sensing [7], cinematography, aerial mapping [8], tracking [9], inspection [10], law enforcement, surveillance [11], search and rescue, and even exploration of the planet Mars [12].

A real-time vision sensor suitable for micro-UAVs needs to overcome weight and power restrictions, and equip micro-UAVs with 1) adequate sensors for it to maintain stable flight and 2) sufficient computational power to process data from those sensors to achieve real-time control. Many current solutions bypass this problem by using ground stations to process data from image sensors on the micro-UAV. This dependency on a ground station for computation of sensors and control not only limits the operating range of a hovering micro-UAV, but its reaction time as well. As bin Ramli et al. [13] note, onboard processing of all sensors related to control of the micro-UAV increases robustness. It can be said that any vehicle that is dependent on a fixed ground station for processing of sensor data required to control the micro-UAV is not a fully autonomous platform.

This article describes the successful implementation of a previously unrealized low-power, light-weight, onboard vision sensor capable of providing sufficient real-time data for a micro-UAV to operate autonomously. First, the article describes the hardware used to realize the real-time vision sensor. The core of the vision sensor hardware is a field programmable gate array (FPGA) because of its ability to exploit the parallelism found in image processing tasks better than central processing units (CPUs) and digital signal processors (DSPs) [14], and their shorter development time and reconfigurability relative to application-specific integrated circuits (ASICs). The section describing the vision sensor hardware also includes a brief description of the target micro-UAV platform, a quad-rotor design referred to as Helio-copter. Following the description of the hardware used, details will be given of the vision algorithms implemented

for the vision sensor. The first set of vision algorithms is intended to correct drift of hovering micro-UAVs and include Harris feature detection, template matching feature correlation, and a similarity-constrained homography estimation using the RANdom SAmple Consensus (RANSAC) algorithm. The second set of vision algorithms is intended to allow a hovering micro-UAV to change from holding a fixed position to tracking a specific object of interest. This set of algorithms includes color segmentation, radial distortion correction, and a Kalman filter that uses a standard-deviation outlier rejection technique (SORT). Performance results of the implementation of these two sets of vision algorithms are then presented, including some preliminary empirical results of the Helio-copter platform detecting and correcting drift, and tracking a target consisting of colored dots.

II. Related Work

Current hovering micro-UAV solutions using vision sensors either 1) transmit images from onboard image sensors to offboard processing platforms (e.g. [15–17]), or 2) they include offboard fixed image sensors and processors (e.g. [18,19]). Both types of systems require images to be sent to a ground station where various vision algorithms are performed from which control commands are generated and sent back to the micro-UAV. Vision algorithms are computationally intensive, requiring a lot of processing power. This is the reason why most current micro-UAVs perform these computations on a ground station where large processors and high power requirements can be met [20]. The need to transmit images and commands to a ground station limits the range of the aircraft to the range of the wireless technology used. The transmission of images also introduces other problems, like noise and delay. Corrupted wireless transmissions are common and can reduce the image quality for analog transmissions and slow down the transfer rate for digital transmissions. Delays in receiving commands from vision sensors also occur because of the time it takes to transmit images to the ground, process them, and transmit resulting commands back.

There are many publications describing efforts to provide adequate sensor data to allow control of hovering micro-UAVs. Altüg et al. [10] used a fixed ground camera to estimate position and orientation of a quad-rotor to aid in achieving stable flight. Their work was later extended to use both the ground camera and a camera onboard the quad-rotor transmitting images to a ground station computer to achieve more autonomy [18]. Earl and D'Andrea [19] achieved successful attitude estimation results for a quad-rotor by decomposing onboard rate gyro measurements and offboard vision sensor measurements and then applying a Kalman filter to them. An actuated two-degree-of-freedom camera to help control the quad-rotor for tracking targets was successfully simulated by Neff et al. [15]. A vision-based obstacle avoidance algorithm for micro-UAVs is presented by Zufferey and Floreano [17]. A simple vision sensor using offboard computation hardware is proposed by Romero et al. [21] to overcome the challenges of flying a quad-rotor indoors. Chitrakaran et al. [16] describe an error-bounded controller to land a quad-rotor, assuming a monocular onboard camera to measure position information, using Lyapunov design methods.

Hirai et al. [4] implemented three vision algorithms to compute the image gravity center, object orientation detection using radial projection, and Hough transform on an FPGA-based system. The FPGA-based LSAVision architecture with Boavista sensor and their application to docking autonomous surface and underwater vehicles is described by Martins et al. [3]. Silva et al. [2] describe the application of the same system on a ground vehicle in the Robocup competitions. Real-time vision systems have been implemented onboard large UAV platforms, providing feedback to allow them to land on designated landing pads [22,23]. All of these vision-sensor-based solutions require a ground station to process the image information, or use platforms that are large enough to not have weight and power restrictions. A smaller FPGA-based vision system described by Kjaer-Nielsen et al. [1] uses separate IEEE 1394 cameras, and performs image preprocessing tasks similar to those implemented in the first stage of the Helio-copter vision sensor.

III. Vision System Hardware

To accomplish real-time image processing on a micro-UAV, the right combination of light-weight, low-power hardware is required. An FPGA-based solution was considered because of the possibility of obtaining a light-weight, low-power vision-sensor that has strengths in real-time computing through pipelined and parallel processing. Implementing vision algorithms on FPGAs not only has the potential to increase throughput over a software implementation, but if it were light-weight and low-power, it would allow onboard placement, removing the transmission-to-decision

latency of using a ground station from the system. If images can be processed onboard the micro-UAV, noise will not be introduced into the images during transmission, which increases image quality and improves results.

A compact FPGA board designed for onboard image processing on small autonomous vehicles called Helios [24] was used as the base of the vision sensor. It is approximately the size of a deck of cards, weighs less than 50 g, and in typical configurations consumes 1–3 W of power. In addition to these desirable power and weight specifications, the Helios board also contains hardware useful for image processing tasks, such as SDRAM and SRAM memory. Helios also has a 120 pin header that allows it to connect to daughter boards built to include less general, more application specific hardware. The version of Helios used for this work contained a Virtex-4 FX60 FPGA that is equipped with two embedded microprocessors and over 56,000 configurable logic blocks.

A daughterboard incorporating hardware necessary for this work was developed and combined with a Helios board. This board is referred to as the autonomous vehicle toolkit (AVT) daughterboard. The AVT board has connections for up to two CMOS image sensors, and contains a smaller FPGA meant to preprocess images to a standard RGB image format before they are relayed to the FPGA on Helios. The AVT also contains a ZigBee wireless modem for wireless communication of up to one mile, and serial ports to communicate with micro-UAV control hardware.

A. Hovering Micro-UAV Platform

In order to empirically test the completed vision sensor, a hovering micro-UAV platform was designed and built. Design specifications were set to require the desired platform to have a total payload capacity of 5 lb at the current elevation of the testing location of 4,500 ft and achieve a flight time of 30 min. A quad-rotor design was selected and was built from mostly off-the-shelf components resulting in the platform shown in Fig. 1. The commercially available Kestrel Autopilot [25] was used to control the quad-rotor. It consists of an inertial measurement unit (IMU) integrated with a 29 MHz Rabbit processor and receives data from the vision sensor through serial communication.

IV. Drift Stabilization Algorithms

In order to provide image processing solutions onboard micro-UAVs, a balanced combination of software and hardware resources was used to implement selected algorithms. This section describes an onboard image processing solution to detect and track features that are used to estimate the movement or homography from one frame to the next for the Helio-copter. First, features are detected in each image using the Harris feature algorithm and correlated using a template matching algorithm. The resulting corresponded features are used to estimate a similarity-constrained homography using a RANSAC algorithm. Second, a target tracking algorithm is explained that uses color segmentation to identify two points on a target. Other algorithms are then used to correct radial distortion, reject outliers, and estimate missing point locations. The resulting filtered points are also used to estimate a similarity-constrained homography between the Helio-copter and the target. The following sections describe details of each algorithm used to address each of these two capabilities of the Helio-copter vision sensor.

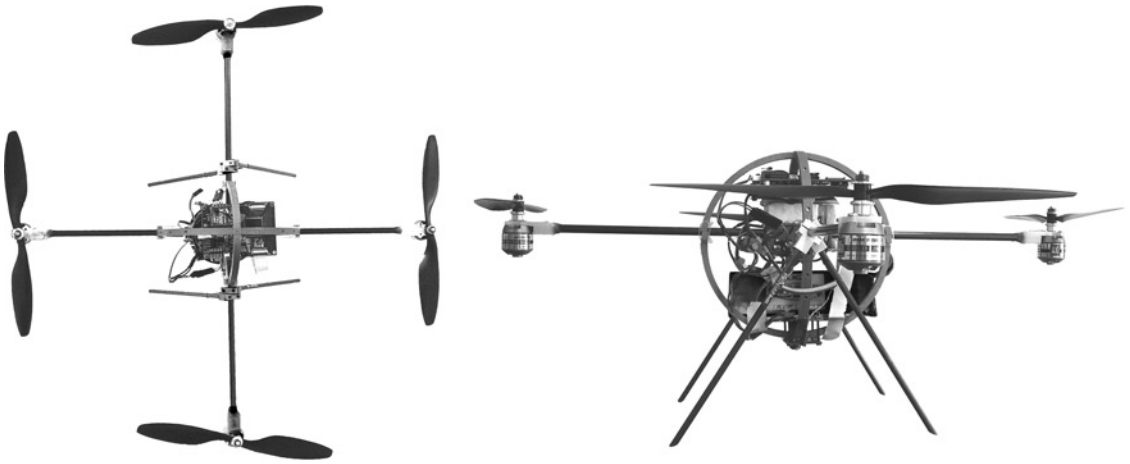

Fig. 1 Top and side views of Helio-copter platform.

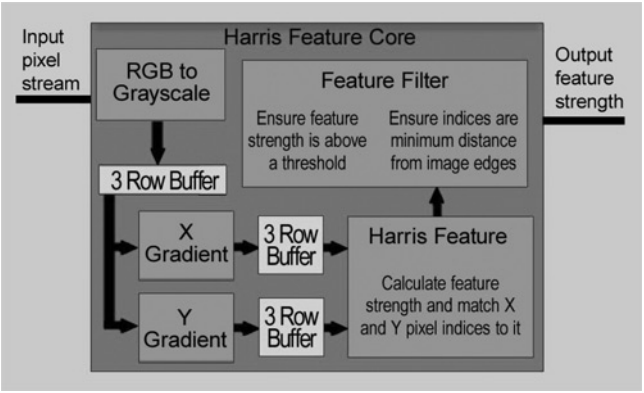

Fig. 2 Block diagram of data flow through Harris feature core.

A. Harris Feature Detector Implementation

The Harris feature detection algorithm was selected for its robustness and repeatability over sequences of images modified by rotation, warping, scaling, lighting, etc. [26]. To implement the Harris feature detector algorithm in a pipelined format required the use of buffers that could accumulate three rows of grayscale image data which streams at one pixel per clock cycle from the image sensor. Besides the buffers, the algorithm was also broken into X and Y gradient blocks, and a Harris feature block as can be seen in the data flow diagram in Fig. 2.

After the Harris feature algorithm was partitioned, it was also modified as is necessary when adapting most algorithms for implementation in hardware. These few modifications were made to the original algorithm to use integer operations instead of floating-point operations. During the evaluation of Eqs (1) and (A), three signed numbers are produced when pixel intensities $I(x, y)$ are unsigned 8-bit values. The sum of these three numbers will require more than 8 bits to avoid overflow issues. Since more bits are already required and only the magnitude of each of the three intermediate values is important, FPGA DSP multipliers were used to square these values to remove the sign. The three resulting 16-bit values are then shifted down 4 bits, summed together, and stored as 12-bit numbers which can be multiplied according to Eq. (3). This allows the determinant of G to fit in 24 bits and the trace of G to fit in 26 bits. The resulting Harris Feature value was truncated to 13 bits so that, when concatenated with the 10-bit x index and 9-bit y index, it results in a 32-bit value. This truncation introduced an automatic threshold of feature strengths.

$$\begin{aligned}\nabla I_v(x, y) = {} & I(x+1, y-1) - I(x-1, y-1) + I(x+1, y) - I(x-1, y) \\ & + I(x+1, y+1) - I(x-1, y+1) \end{aligned} \tag{1}$$

$$\begin{aligned}\nabla I_h(x, y) = {} & I(x-1, y-1) - I(x-1, y+1) + I(x, y-1) - I(x, y+1) \\ & + I(x+1, y-1) - I(x+1, y+1) \end{aligned} \tag{2}$$

$$G = \begin{bmatrix} \nabla I_v^2 & \nabla I_v \nabla I_h \\ \nabla I_v \nabla I_h & \nabla I_h^2 \end{bmatrix} \tag{3}$$

B. Feature Correlation

The features found by the Harris feature detector are correlated between images using a template matching hardware core. The template matching feature correlator core uses a template of 8×8 pixels around each feature given by the Harris feature detector core to search for the same feature within a 32×64 window in the subsequent image. This core is made up of four different types of hardware blocks, one block to manage image addresses in memory, a block to interface to SRAM, blocks to calculate sum of squared differences (SSD) between the template and the search area called processing elements, and a block to dispatch templates and search areas to the processing elements. Once the template matched features are correlated then the list of paired feature indices is stored in SRAM and an interrupt is asserted to signal the RANSAC homography software to start processing the data. A diagram showing the flow of data between each of these blocks is shown in Fig. 3.

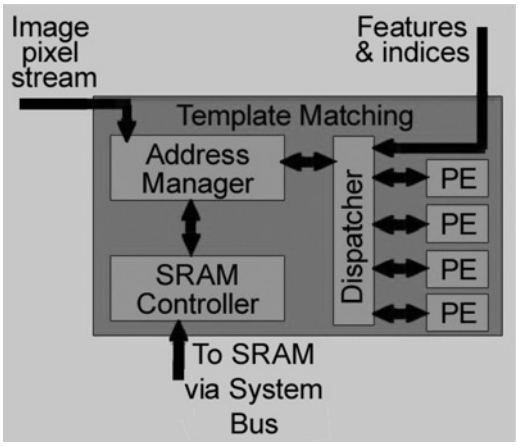

Fig. 3 Block diagram of template matching core.

1. Image Address Manager

The template matching correlator core receives the current grayscale image pixel stream and stores it in SRAM. The 4 MB SRAM that is available on the Helios board was used to store up to eight images for this core. The image address manager block in this core rotates through the SRAM keeping track of the address of the current image and the previous image. It also tracks the status of the current image, whether or not it has been completely written, so that it can signal the template matching process to start.

2. SRAM Controller

Connected to the image address manager is a dual-port SRAM controller. It was implemented to handle image data transfer to and from the external SRAM on Helios. The controller takes advantage of a three-stage pipeline to hide the three-cycle latency of the SRAM, so that read or write requests are issued on every clock cycle. This allows the full bandwidth of the SRAM to be available, minimizing the affect that the bottleneck of memory accesses has on the system.

3. Dispatcher

Also connected to the image address manager is the dispatcher. It receives feature indices (x, y) from the Harris feature block and sends a request to the address manager for a template area from one image and a search area from the subsequent image, and then assigns a processing element to compare the areas. The processing elements require a set amount of time to complete execution and do not lend themselves to a pipelined architecture. This causes the dispatcher to have idle periods waiting for a processing element to become available. Feature locations from the Harris feature block are buffered by the dispatcher, allowing it to keep all the processing elements fully utilized without losing incoming features.

4. Processing Elements

The dispatcher controls multiple processing elements. The processing element blocks perform all the template matching calculations. Each of the processing elements use an FPGA Block RAM (BRAM) for template and search area storage. Each element calculates a SSD between the 8×8 template and 8×8 windows in the search area. The 8×8 window in the search area that has the smallest SSD with the template is considered a match and the center x and y indices are stored with the original features x and y indices as a feature correlated pair.

The processing elements are assigned search areas to process in groups of four due to the nature of the memory access bandwidth of the FPGA BRAMs. There are two read/write ports for each BRAM that are each 4 B wide. This means a maximum of 8 B can be read per clock cycle, and since the templates are 8 B wide, a whole row of the template can be calculated at the same time in parallel. To take advantage of this, four processing elements are

each given a byte-shifted copy of the same search area, allowing a full search area for one feature to be completed in about 3,000 clock cycles or 30 μs.

5. Algorithm Modifications for Hardware

The image address manager, SRAM controller, dispatcher, and processing elements together make up the complete template matching hardware core. As these were implemented, modifications were made to the template matching algorithm to maximize efficient use of FPGA resources. One modification that was made concerned the size of the template used to estimate how much a feature matches. Standard approaches use a symmetric number of pixels on all sides of a feature location, which results in odd-value-sized templates, e.g. 5×5, 9×9. FPGA storage resources required to implement a template matching algorithm, like BRAMs, have even, multiple-of-two sized ports. To maximize efficient use of the read and write operations for these resources, an asymmetric 8×8 pixel template is used instead of the standard symmetric templates. This allows a full template to be written to two 4 B ports of an FPGA BRAM in 16 write operations instead of 24 write operations like a 9×9 template would require.

Another modification made to the algorithm involved changing the search area size to increase efficient use of the FPGA resources. Traditionally the constraints on the size of the template matching search area has been determined by available CPU processing capabilities. Because so many search areas can be processed in parallel on the FPGA, the concern shifts away from reducing processing time to being able to fit the search area into the local BRAM storage. BRAM sizes on the Helios FPGA are 2048 B which allowed for a rectangular search window of 64×32. Although a square search window of 45×45 would also fit in the BRAMs and allow for uniform searching in all directions from the original feature location, the 64×32 search window was chosen to reduce control logic needed to address the search area using the BRAM 9-bit address. The top 5 bits of the address were used to address each of the 32 rows of the search area, while the bottom 4 bits increments through sixteen 4 B words of each row.

C. RANSAC Similarity-Constrained Homography Estimation

Once features are identified by the Harris feature detector core and correlated across images by the template matching core, a RANSAC similarity-constrained homography algorithm described by Johansen [27] was modified and used to determine the translation, rotation, and scaling difference between the images.

The similarity-constrained homography equations used in the RANSAC algorithm are derived from the similarity model shown in Eq. (4), where s, θ, T_x, and T_y are the scale factor, angle of rotation, translation along the x axis, and translation along the y axis, respectively.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = s \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix} \tag{4}$$

Although the RANSAC similarity-constrained homography algorithm was implemented, modifications were made to ensure real-time performance. The original algorithm suggested determining a similarity-constrained homography using a least squares approach, but this was changed to calculate an exact homography solution using only two feature points. Each of the other feature points (x, y) was then transformed using this homography and the Euclidean distance between the result and the matched feature point (x', y') was thresholded to vote for or against the homography. After a specific number of iterations the similarity-constrained homography with the most votes was returned. Using one pair of points reduces the amount of computation needed to generate a similarity-constrained homography by reducing an over-constrained problem of multiple points to a problem with an exact solution.

V. Target Tracking Algorithms

Algorithms were also implemented for the vision sensor that would allow the Helio-copter to maintain stable attitude while tracking a target. To maintain a stable position over a fixed scene, features throughout the whole image need to be used to correctly estimate homographies between images. To follow a moving target requires the image processing hardware to ignore the translation of all features in the image except those defined as being the desired target. For practical applications a quad-rotor would need to be able to hold a position over a scene until a user selected a target in the image scene at which point the vision sensor would switch tasks and use features of the target to hold its position, moving if the target moved. As a proof-of-concept of the ability of the vision sensor

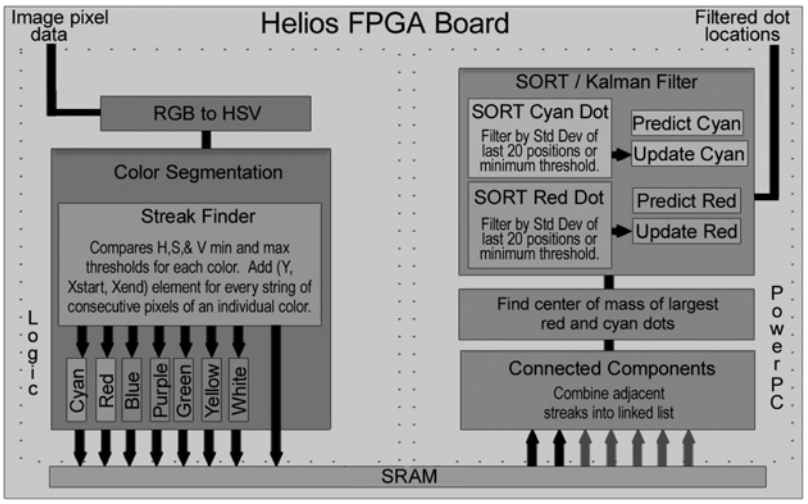

Fig. 4 Data flow through components of the target tracking system.

of the Helio-copter to perform a target tracking task, a target of two colored dots was used. This controlled test target allowed a display of the capabilities of the vision sensor by implementing color space conversion, a color segmentation algorithm, connected components algorithm, radial distortion correction, and a Kalman filter using a SORT. A block diagram of this system is shown in Fig. 4.

A. Color Segmentation and Connected Components

The first algorithms implemented for the target tracking component of the vision sensor was a color segmentation and connected component analysis algorithms. The color segmentation FPGA core includes an RGB to HSV color space conversion block, a hue, saturation, and intensity threshold block, and a block that identifies streaks of contiguous pixels having the same color. Structures containing 'streak' information are stored in memory and then connected and stored in a linked list in software. These linked lists are then used to find the center of mass of each of the colored connected components.

Images were converted from the RGB to HSV color space to improve robustness under varying lighting conditions [28]. The HSV color space conversion was pipelined into five stages in the FPGA hardware. Once each pixel has been converted to the HSV color space, it is compared against upper and lower thresholds for hue, saturation, and intensity. This thresholded stream of pixel data is then passed on to the streak finder core as binary values.

The thresholded pixel data from the segmentation block is used in the streak finder block to identify continuous segments of the same color in a row of the image. When the first pixel marked as having the desired color is found, both the row and column indices are stored in registers. These indices are saved if subsequent pixels of the same color are received. As soon as a pixel is received that is not of the same color or the end of the row is reached, then the core combines the registered row and column indices and the current column index referred to as streaks and stores them in memory.

After finding the streaks in each row of an image, a software routine groups together the streaks in linked list structures to form connected components. A streak is added to a linked list if it lies in an adjacent row and has overlapping starting and ending column indices of another streak in the list. If no list with an adjacent streak is found then a new list is created.

As linked lists of streaks are created, minimum and maximum row and column indices are kept so that the index of the center of mass (x_c, y_c) can be computed. Since the connected components used in this project are circles, the center of mass can be found using the equations in (5). The number of pixels in each streak in a connected component

is also accumulated and stored as its mass, which can be thresholded for noise rejection.

$$\begin{aligned} x_c &= (x_{\min} + x_{\max})/2 \\ y_c &= (y_{\min} + y_{\max})/2 \end{aligned} \tag{5}$$

For this project two colored dots, red and cyan, were used to form the target. The center of mass of each connected component was used for the similarity-constrained homography calculation. Using two points is a minimum for a similarity-constrained homography calculation and it was observed that certain cases would cause the system to not detect one of the connected components or misrepresent the locations of the centers of mass. To correct for these cases, three algorithms were implemented and added to the system before the similarity-constrained homography was calculated. These algorithms included a radial distortion correction, an outlier rejector, and a Kalman filter.

B. Correcting Radial Distortion

The first algorithm implemented to improve accuracy of target detection was an algorithm to correct for radial distortion by the camera lens. The lens used on the image sensor for the target tracking task is a wide angle 2.1 mm lens which allows an area of about 6×6 ft to be visible from an altitude of approximately 6 ft. This lens has a wider field of view than larger lenses, but also warps the image towards the edges. The distortion in the image negatively affects the scale factor of the similarity-constrained homography.

One of the difficulties of correcting radial distortion is the lack of a general algebraic expression for the inverse mapping of distorted image points to their undistorted locations [29]. To overcome this, a polynomial equation was used to approximate the undistorted locations of the distorted image points. The polynomial coefficients α_1, α_2, and α_3 from Eq. (7) were found offline by correlating features between distorted and rectified images of a calibration grid. Equations (7)–(10) were then incorporated into the vision sensor to estimate undistorted locations (x_u, y_u) of the distorted points (x_d, y_d). Radial distortion occurs with respect to the center of the image, and the pixel indices are registered in the hardware core with respect to the upper left hand corner of the image. The origin must be shifted before the points are undistorted, Eq. (6), and then returned to the original origin afterward, Eq. (11).

$$\begin{aligned} x_d &= x_d - \frac{W}{2} \\ y_d &= \frac{H}{2} - y_d \end{aligned} \tag{6}$$

$$r = \alpha_1 (x_d^2 + y_d^2) + \alpha_2 \sqrt{x_d^2 + y_d^2} + \alpha_3 \tag{7}$$

$$\theta = \arctan \frac{y_d}{x_d} \tag{8}$$

$$y_u = \begin{cases} r \sin\theta & \text{if } x_d \geqslant 0 \\ -r \sin\theta & \text{if } x_d < 0 \end{cases} \tag{9}$$

$$x_u = \begin{cases} r \cos\theta & \text{if } x_d \geqslant 0 \\ -r \cos\theta & \text{if } x_d < 0 \end{cases} \tag{10}$$

$$\begin{aligned} x_u &= \frac{W}{2} + x_u \\ y_u &= \frac{H}{2} - y_u \end{aligned} \tag{11}$$

C. Standard-Deviation Outlier Rejection Technique

After correcting for radial distortion, calculating a similarity-constrained homography using two points proved sufficient except when one of the points was not detected correctly. This occurred sometimes when the Helio-copter or the target moved quickly enough that a dot left the image before the Helio-copter was able to recover, or an object

temporarily obstructed the view of a dot. To increase the robustness of the target tracking system, a Kalman filter was implemented. Given the fact that a Kalman filter does not in and of itself detect outliers, a SORT was used in conjunction with it.

The SORT algorithm stores the previous 20 locations of both the red and cyan dots. The standard deviation of each set of 20 points is calculated and used to threshold the newest dot locations. As mentioned, the range method uses the mean $\pm\, n \times$ stdev as the threshold to label outliers. A value of $n = 3$ was found empirically to work well. Through experimental results it was determined that the magnitude of $n \times$ stdev also needed to be saturated to a fixed lower limit to prevent it from getting too small if the platform did not move much.

D. Kalman Filter

A Kalman filter is able to use a basic physics model to estimate the center of mass of a dot from its previous location when it is not detected correctly. The Kalman filter ends up smoothing out drastic changes in the locations of the dots, preventing drastic control commands that could cause the Helio-copter to become unstable, while still allowing it to follow the target.

In this implementation, when a point has been classified as an outlier, the Kalman filter responds by modifying the sensor update covariance matrix so as to be less confident in that point. An extended Kalman filter was implemented independently for each dot, with separate prediction and updating stages for the red and cyan dots, as shown in Fig. 4. Each filter's state vector (X_k) was composed of the x and y positions of the dot it was filtering (Eq. (12)). The sensor input (Z_k) to the filter is also the x and y positions of each dot, making the H matrix of the Kalman filter update equations in (13) the identity matrix.

$$T_j^{\text{color}} = \begin{bmatrix} x^{\text{color}} \\ y^{\text{color}} \end{bmatrix} \tag{12}$$

$$\begin{aligned} \hat{U}_j &= Z_j - H_j \hat{T}_{j|j+1} \\ S_j &= H_j P_{j|j-1} H_j^T + R_j \\ K_j &= P_{j|j-1} H_j^T S_j^{-1} \\ \hat{T}_{j|j} &= \hat{T}_{j|j-1} + K_j \hat{U}_j \\ P_{j|j} &= (I - K_j H_j) P_{j|j-1} \end{aligned} \tag{13}$$

The equations are then simplified even further by breaking all matrix operations into single element operations shown in Eq. (14).

$$\begin{aligned} \hat{u}(1)_j &= z(1)_j - \hat{t}(1)_{j|j+1} \\ \hat{u}(2)_j &= z(2)_j - \hat{t}(2)_{j|j+1} \\ s(1)_j &= p(1,1)_{j|j-1} + r(1)_j \\ s(2)_j &= p(2,2)_{j|j-1} + r(2)_j \\ k(1)_j &= \frac{p(1,1)_{j|j-1}}{s(1)_j} \\ k(2)_j &= \frac{p(2,2)_{j|j-1}}{s(2)_j} \\ \hat{t}(1)_{j|j} &= \hat{t}(1)_{j|j-1} + k(1)_j \hat{u}(1)_j \\ \hat{t}(2)_{j|j} &= \hat{t}(2)_{j|j-1} + k(2)_j \hat{u}(2)_j \\ p(1,1)_{j|j} &= p(1,1)_{j|j-1} - k(1)_j p(1,1)_{j|j-1} \\ p(2,2)_{j|j} &= p(2,2)_{j|j-1} - k(2)_j p(2,2)_{j|j-1} \end{aligned} \tag{14}$$

The predict stage function for the cyan dot is shown in Eq. (16). This part of the Kalman filter is what allows the Helio-copter to track the target for short times when one of the dots is not detected. Normally the filter predicts the position of the dot using the velocity of the dot calculated from its current position and its position four frames prior (Eq. (15)), but when a dot is not detected, then it will use the velocity of the other dot. The success of this algorithm is based on the assumption that if the Helio-copter loses a dot, it will not drastically change altitude or heading before the dot is recovered. Experimental results are discussed to support the validity of this assumption.

$$\delta(\tau)_j = \frac{\tau_j - \tau_{j-4}}{4} \tag{15}$$

$$\begin{bmatrix} x_j^{\text{cyan}} \\ y_j^{\text{cyan}} \end{bmatrix} = \begin{cases} \begin{bmatrix} x_{j-1}^{\text{cyan}} + \delta(x)_{j-1}^{\text{cyan}} \\ y_{j-1}^{\text{cyan}} + \delta(y)_{j-1}^{\text{cyan}} \end{bmatrix} & \text{if dot}^{\text{cyan}} \notin \text{ outlier} \\ \begin{bmatrix} x_{j-1}^{\text{cyan}} + \delta(x)_{j-1}^{\text{red}} \\ y_{j-1}^{\text{cyan}} + \delta(y)_{j-1}^{\text{red}} \end{bmatrix} & \text{otherwise} \end{cases} \tag{16}$$

VI. Results

The processing capabilities of the completed vision sensor included simultaneously computing and storing ten sequential 640 × 480 images of each of the following: an original RGB image, the HSV conversion of the original, eight color-segmented images, a rank transform image, a Harris feature image, and template matched feature correspondences between each pair of Harris feature images. The vision sensor was able to process all these computations and achieve frame rates over 37 frames per second all while occupying only about 1/3 of the logic blocks on the Helios FPGA. Qualitative results for each of the implemented vision algorithms are given in the following sections.

A. Feature Detection and Correlation

Performance results of the Harris feature detector and template matching correlation hardware showed that as many as 2,636 features per frame could be detected and matched at 30 frames per second. Using template matching for feature correlation guarantees the features with the minimum summed squared difference will be found within the search window. The fact that the algorithm is able to operate with a real-time frame rate indicates a high likelihood that features will not move farther than search window boundaries in subsequent frames.

B. RANSAC Similarity-Constrained Homography

To test the RANSAC similarity-constrained homography algorithm, three sequences of images and resulting feature points were captured of controlled scene movements. These three sequences included a sequence with the scene rotating tangentially to the image plane, another with the scene translating horizontally along the image plane, and the

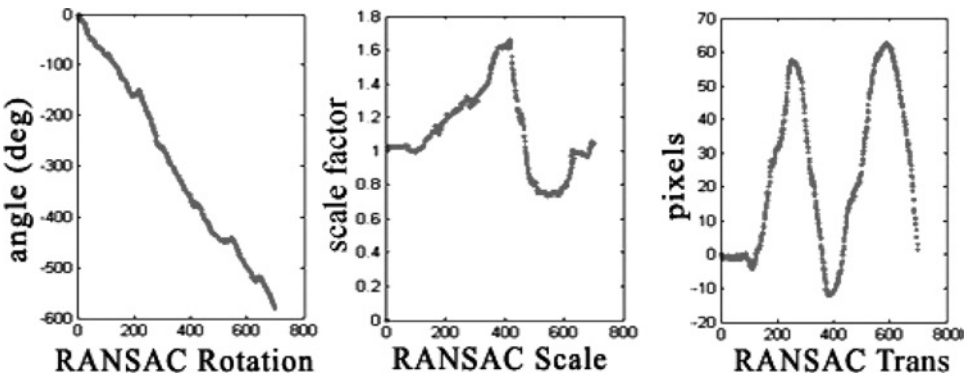

Fig. 5 RANSAC similarity-constrained homography test results; graphs are results of rotation, scale, and *x*-axis translation tests.

last with the scene translating transversely to the image plane to decompose the three elements, rotation, translation, and scale factor, of a similarity-constrained homography. The similarity-constrained homography estimations for each pair of images in the sequences were accumulated in order to compare estimated scene movement with the actual movement over the entire sequence. Figure 5 shows the three plots of the RANSAC similarity-constrained homography algorithm results. For the rotation test, the scene was rotated 636 deg in a series of revolutions. The RANSAC algorithm accumulated a measurement of 596 deg, resulting in an accumulated error of 6.29%. The scene was moved toward the camera, away from it, and then back towards it for the test involving just changes in scale for a total accumulated change in scale factor of 4.08. The algorithm estimated a total of 4.20, resulting in an error of 2.69%. The translation-only test involved moving the scene from left to right two times for a total distance of 225 pixels, which resulted in an error of 2% with the RANSAC estimation of 220 pixels.

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

C. Color Segmentation and Connected Components

Performing color space conversion and color segmentation in hardware takes advantage of the ability to segment as many colors as needed simultaneously, all while only adding a few more stages to the image pipeline. Implementing the streak finder in hardware over software reduces hundreds of milliseconds of processing time to just a few extra clock cycles of latency. This reduces a potential delay of as much as half a second for a software implementation to less than 20 clock cycles of added latency in hardware, which at the current speed of 300 MHz is less than 4 ns. Creating the linked lists for the connected components is more efficient when implemented in software than hardware because of the multiple data pointers to memory. Although the measured frame rate of the color segmentation and connected components processes varied slightly in the range 35–45 fps depending on color content of the images, it was sufficient to handle the set 30 fps output frame rate of the image sensor.

D. Radial Distortion Correction

Empirical tests involving the Helio-copter were performed to obtain results of the radial distortion correction. Flight tests were performed where the Helio-copter was manually translated over the two-dot target. Without radial distortion correction, the Helio-copter experienced an altitude change of approximately 6–8 in. as the target moved from the center to the edge of the image. These changes in altitude became negligible when repeating the same tests with the radial distortion correction incorporated into the vision sensor.

E. Kalman Filter and SORT

The Kalman filter was first simulated using actual positions and velocities of the target dots logged during a Helio-copter test flight. After achieving successful simulation results, the filter was implemented on the vision sensor. Once again, dot positions were logged, this time with the Kalman filtered points as well. The logged values were taken from a test where one of the dots leaves the image for a period of time and re-enters at a different point on the image. The logged data for one of the two dots was then plotted using triangles to denote observed dot positions from the image sensor, and crosses to denote Kalman filter estimates. Figure 6 shows this plot. As can be seen, the estimated locations of the dot are nearly centered on the observed locations of the dot starting at point A continuing through a full circle until reaching point C. At point C the dot disappeared from the view of the camera and therefore no valid observed positions were logged. The dot remained out of view of the camera until re-entering at point D. Between points C and D, the Kalman filter continued to estimate the location of the missing dot using the velocity vector of the visible dot. This is reflected on the plot in Fig. 6 by the path of crosses without corresponding triangles between points C and D. When the dot re-entered the view of the image sensor at point D, the Kalman filter estimations are within a dozen pixels of the actual location of the dot.

Various tests were performed to evaluate the effectiveness of SORT in detecting outliers. A sample case where the target moved very abruptly caused the algorithm to reject the large change at first, but when enough measurements verified the new location, the standard deviation increased until the new location was not considered an outlier and the filter-estimated dot positions followed a smooth yet quick change to update to the new location. The filter also showed

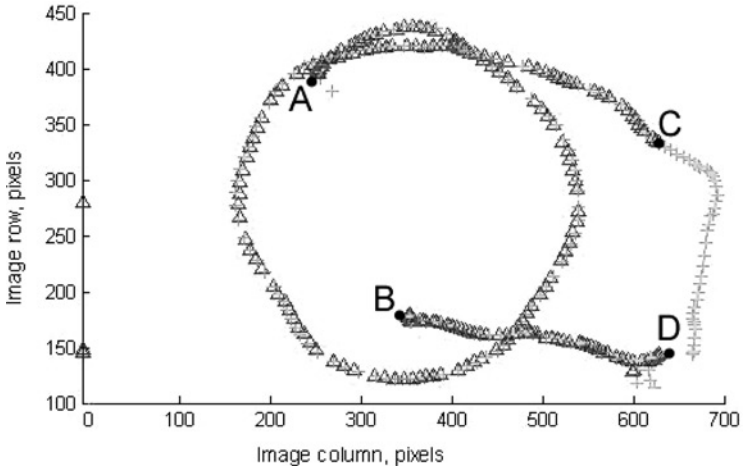

Fig. 6 Kalman filter performance in the case where a dot is lost from the image (section of plot from point C to point D).

a resistance to large accelerations and random jumps as expected. The implementation of the extended Kalman filter with SORT significantly increased the robustness of the target tracking component of the vision sensor.

F. Helio-copter Drift Stabilization and Target Tracking

As a final proof of concept, the completed vision sensor was installed on the Helio-copter platform and tested. Comparison flight tests of the Helio-copter with and without the vision sensors activated, while maintaining all control parameters constant, provided empirical proof of concept. Without the vision sensor the Helio-copter was not able to detect nor correct drift or track a target. With the vision sensor activated, the Helio-copter visibly corrected its drift continually holding a fixed position within a 6×6 ft area for up to 43 s. The stability and performance of the Helio-copter during these tests varied as the control model and parameters were modified, but tuning them so that the Helio-copter could hold a fixed position indefinitely remains a challenging task for researchers interested in more robust control theories. The Helio-copter also achieved similar performance while tracking the desired target.

VII. Conclusion

The implementation of the specific vision algorithms described on the Helios and AVT hardware resulted in a previously unrealized low-power, light-weight, onboard vision sensor capable of providing sufficient real-time data for a micro-UAV to operate autonomously. Quite a few of the vision sensor's available resources were not even used. Only about 1/3 of the current Helios FPGA slices are utilized even though the synthesis of the design was not configured to optimize hardware usage, which allowed circuit routing to use far more slices than needed. Even more programmable logic is available if unused resources on the Spartan FPGA on the AVT daughterboard are considered. The low use of computational resources for this project indicate that this vision sensor has great potential for implementing additional vision algorithms.

Hardware/Software co-designs of a Harris feature detector, a template matching feature correlator, a RANSAC homography algorithm, color space conversion, a color segmentation algorithm, a connected components algorithm, radial distortion correction, and a Kalman filter using a SORT were implemented achieving real-time performance without the aid of a processing ground station. This success has encouraged the plans of future work to include development and implementation of stereo vision algorithms to add to this vision sensor.

References

[1] Kjaer-Nielsen, A., Jensen, L. B., Sørensen, A. S., and Krüger, N., "A Real-Time Embedded System for Stereo Vision Preprocessing Using an FPGA," *Proceedings of the International Conference on Reconfigurable Computing and FPGAs*, IEEE Computer Society, Los Alamitos, CA, 2008, pp. 37–42.
doi: 10.1109/ReConFig.2008.63

[2] Silva, H., Almeida, J. M., Lima, L., Martins, A., and Silva, E. P., "A Real Time Vision System for Autonomous Systems: Characterization During a Middle Size Match," *RoboCup 2007: Robot Soccer World Cup XI*, Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 504–511.

[3] Martins, A., Almeida, J. M., Ferreira, H., Silva, H., Dias, N., Dias, A., Almeida, C., and Silva, E. P., "Autonomous Surface Vehicle Docking Manoeuvre with Visual Information," *Proceedings of the IEEE International Conference on Robotics and Automation*, IEEE Robotics and Automation Society, Roma, Italy, 2007, pp. 4994–4999.
doi: 10.1109/ROBOT.2007.364249

[4] Hirai, S., Zakoji, M., Masubuchi, A., and Tsuboi, T., "Realtime FPGA-Based Vision System," *Journal of Robotics and Mechatronics*, Vol. 17, No. 4, 2005.

[5] Ettinger, S. M., Nechyba, M. C., Ifju, P. G., and Waszak, M., "Vision-Guided Flight Stability and Control for Micro Air Vehicles," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and System*, Vol. 3, IEEE Robotics and Automation Society, EPFL Lausanne, Switzerland, 30 Sept.–5 Oct. 2002, pp. 2134–2140.
doi: 10.1109/IRDS.2002.1041582

[6] Barrows, G. L., "Future Visual Microsensors for Mini/Micro-UAV Applications," *Proceedings of the 2002 7th IEEE International Workshop on Cellular Neural Networks and their Applications* (*CNNA 2002*), World Scientific Publishing, Frankfurt, Germany, 22–24 July 2002, pp. 498–506.
doi: 10.1109/CNNA.2002.1035087

[7] Archer, F., Shutko, A. M., Coleman, T. L., Haldin, A., Novichikhin, E., and Sidorov, I., "Introduction, Overview, and Status of the Microwave Autonomous Copter System (MACS)," *Proceedings of the 2004 IEEE International Geoscience and Remote Sensing Symposium, IGARSS'04*, Vol. 5, IEEE, Anchorage, AK, 20–24 Sept. 2004, pp. 3574–3576.

[8] Sugiura, R., Fukagawa, T., Noguchi, N., Ishii, K., Shibata, Y., and Toriyama, K., "Field Information System Using an Agricultural Helicopter Towards Precision Farming," *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM 2003*, Vol. 2, IEEE Industrial Electronics Society, Port Island, Kobe, Japan, 20–24 July 2003, pp. 1073–1078.
doi: 10.1109/AIM.2003.1225491

[9] Rock, S. M., Frew, E. W., Jones, H., LeMaster, E. A., and Woodley, B. R., "Combined CDGPS and Vision-Based Control of a Small Autonomous Helicopter," *Proceedings of the American Control Conference*, Vol. 2, American Automatic Control Council, Philadelphia, PA, 24–26 June 1998, pp. 694–698.

[10] Altüg, E., Ostrowski, J. P., and Mahony, R., "Control of a Quadrotor Helicopter Using Visual Feedback," *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'02*, Vol. 1, IEEE Robotics and Automation Society, Washington, DC, 11–15 May 2002, pp. 72–77.

[11] Jun, L., Shaorong, X., Zhenbang, G., and Jinjun, R., "Subminiature Unmanned Surveillance Aircraft and its Ground Control Station for Security," *Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics*, IEEE, Kobe, Japan, 6–9 June 2005, pp. 116–119.

[12] Patel, C. A., "Building a Testbed for Mini Quadrotor Unmanned Aerial Vehicle with Protective Shroud," Master's Thesis, Dept. of Mechanical Engineering, Wichita State Univ., May 2006.

[13] bin Ramli, M. A., Wei, C. K., and Leng, G., "Design and Development of an Indoor UAV," *RSAF Aerospace Technology Seminar*, Government of Singapore, Air Force School, Singapore, 2007.

[14] MacLean, W. J., "An Evaluation of the Suitability of FPGAs for Embedded Vision Systems," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 3, IEEE Computer Society, San Diego, CA, 20–26 June 2005, p. 131.

[15] Neff, A. E., Lee, D. B., Chitrakaran, V. K., Dawson, D. M., and Burg, T. C., "Velocity Control for a Quad-rotor UAV Fly-by-Camera Interface," *Proceedings of the IEEE SoutheastCon*, IEEE, Richmond, VA, 22–25 March 2007, pp. 273–278.

[16] Chitrakaran, V. K., Dawson, D. M., Chen, J., and Feemster, M., "Vision Assisted Autonomous Landing of an Unmanned Aerial Vehicle," *Proceedings of the 44th IEEE Conference on Decision and Control and 2005 European Control Conference, CDC-ECC'05*, IEEE, Seville, Spain, 12–15 Dec. 2005, pp. 1465–1470.

[17] Zufferey, J.-C., and Floreano, D., "'Toward 30-gram Autonomous Indoor Aircraft: Vision-Based Obstacle Avoidance and Altitude Control," *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA'05*, IEEE, Barcelona, Spain, 18–22 April 2005, pp. 2594–2599.

[18] Altüg, E., Ostrowski, J. P., and Taylor, C. J., "Quadrotor Control Using Dual Camera Visual Feedback," *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'03*, Vol. 3, IEEE Robotics and Automation Society, Barcelona, Spain, 14–19 Sept. 2003, pp. 4294–4299.

[19] Earl, M. G. and D'Andrea, R., "Real-Time Attitude Estimation Techniques Applied to a Four Rotor Helicopter," *Proceedings of the 43rd IEEE Conference on Decision and Control, CDC*, Vol. 4, IEEE Control Systems Society, Atlantis, Paradise Island, The Bahamas, 14–17 Dec. 2004, pp. 3956–3961.

[20] Roberts, J. M., Corke, P. I., and Buskey, G., "Low-Cost Flight Control System for a Small Autonomous Helicopter," *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'03*, Vol. 1, IEEE Robotics and Automation Society, Taipei, Taiwan, 14–19 Sept. 2003, pp. 546–551.

[21] Romero, H., Benosman, R., and Lozano, R., "Stabilization and Location of a Four Rotor Helicopter Applying Vision," *Proceedings of the American Control Conference*, American Automatic Control Council, Minneapolis, MN, 14–16 June 2006, 6 p.

[22] Shakernia, O., Ma, Y., Koo, T., and Sastry, S., "Landing an Unmanned Air Vehicle: Vision Based Motion Estimation and Nonlinear Control," *Asian Journal of Control*, Vol. 1, No. 3, Sept. 1999, pp. 128–145.

[23] Sharp, C. S., Shakernia, O., and Sastry, S. S., "A Vision System for Landing an Unmanned Aerial Vehicle," *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'01*, Vol. 2, IEEE Robotics and Automation Society, Seoul, Korea, 2001, pp. 1720–1727.
doi: 10.1109/ROBOT.2001.932859

[24] Fife, W. S. and Archibald, J. K., "Reconfigurable On-Board Vision Processing for Small Autonomous Vehicles," *EURASIP Journal on Embedded Systems*, Vol. 2007, 2007, Article ID 80141 (14 pages), doi:10.1155/2007/80141.

[25] Beard, R., Kingston, D., Quigley, M., Snyder, D., Christiansen, R., Johnson, W., McLain, T., and Goodrich, M., "Autonomous Vehicle Technologies for Small Fixed Wing UAVs," *AIAA Journal of Aerospace Computing, Information, and Communication*, Vol. 2, No. 1, Jan. 2005, pp. 92–108.
doi: 10.2514/1.8371

[26] Schmid, C., Mohr, R., and Bauckhage, C., "Evaluation of Interest Point Detectors," *International Journal of Computer Vision*, Vol. 37, No. 2, 2000, pp. 151–172.
doi: 10.1023/A:1008199403446

[27] Johansen, D. L., "Video Stabilization and Object Localization Using Feature Tracking with Small UAV Video," Master's Thesis, Brigham Young Univ., 2006.

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

[29] Heikkila, J., and Silven, O., "A Four-Step Camera Calibration Procedure with Implicit Image Correction," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, San Juan, Puerto Rico, 17–19 June 1997, pp. 1106–1112.
doi: 10.1109/CVPR.1997.609468

Ella Atkins
Associate Editor