by Mariano Lizarraga, Renwick Curry, and Gabriel Elkaim (USA)

Reprogrammable UAV Autopilot System (Part 2)

Testing and Results

In the first part of this series, you were introduced to the SLUGS reprogrammable UAV autopilot system. This article details the system's software and flight test results.

In the first part of this two-part series, we introduced the hardware used in the Santa Cruz Low-cost UAV GNC System (SLUGS), an open-source unmanned aerial vehicle (UAV) autopilot primarily focused on supporting guidance, navigation, and control (GNC) research. In this article, we'll focus on the software architecture and algorithms. We'll also present results from temperature compensation, sensor calibration, and flight tests.

SOFTWARE ARCHITECTURE

The SLUGS has two Microchip Technology dsPIC33 digital signal controllers (DSCs) that serve as its main processing units, running at 40 MIPS. These are interconnected via a SPI at 10 MHz and are called the "sensor" DSC and "control" DSC, respectively. The sensor DSC is in charge of reading the sensor data and converting it into a high-quality estimate of the aircraft position and attitude (3-D orientation). The control DSC interacts with the user through the ground station, generating signals to the UAV's control surfaces to stabilize the aircraft and fly it along the user-designated mission (at its highest level, traversing waypoints at a given altitude and airspeed).

One of the key advancements in the SLUGS autopilot is that the DSCs are programmed directly from MATLAB Simulink using a block and signal flow metaphor for visual programming. This offers several advantages. Most researchers in the GNC field are already familiar with MATLAB and use it to prototype and simulate new algorithms. Complicated algorithms and changes are easy to implement, rapidly increasing the prototype-simulate-test cycle. Lastly, the software can be easily retargeted for better processors without having to rework the low-level, back-end code. While this environment sacrifices some efficiency, any performance issues are made insignificant by the speed with which changes can be made and tested.

The sensor DSC's low-level software architecture (see Figure 1) is a combination of blocks provided by Lubin Kerhuel's dsPIC Simulink Embedded Target (ET) (shown in

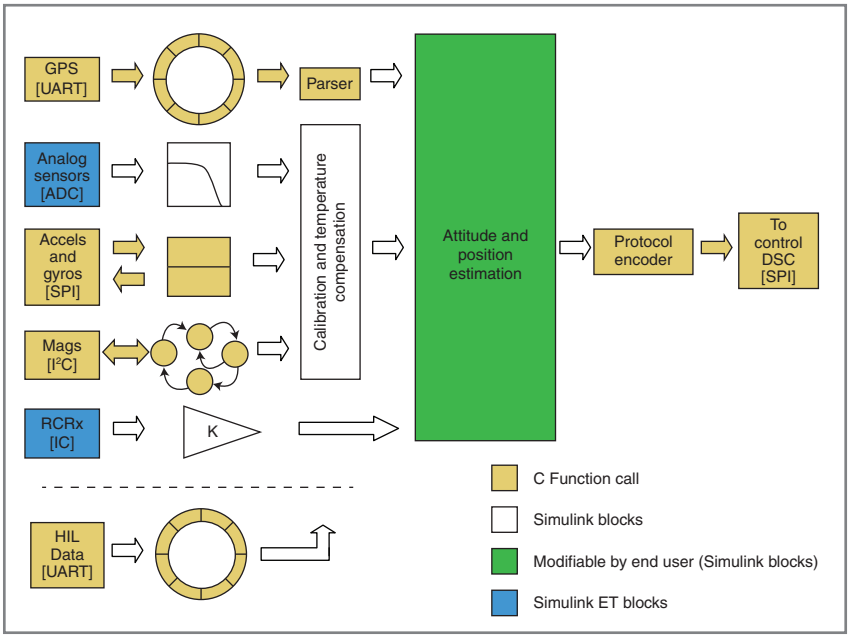

Figure 1—Low-level software architecture for the sensor DSC

blue), blocks implemented in C (shown in yellow), and basic Simulink blocks (shown in white).

Although the ET offered ready-to-use blocks for most of the DSC's peripherals, it was decided early in the design process that many of these would be rewritten in C to provide greater flexibility and full control of the implementation. For instance, to use less processor time while sending data through the serial port, it was necessary to use the direct memory access (DMA) feature of the DSC. The ET's UART block did not offer this capability, and it was necessary to rewrite it in C and provide it as a C function call block in Simulink. This is one of the key advantages of the ET: C functions can be merged into the overall generated code with great ease.

The GPS serial stream is read via a UART port and placed into a circular queue data structure. An NMEA 0183 protocol parser reads data from the circular queue and decodes the protocol sentences producing position, velocities, and course over ground (COG) information. The analog sensors (dynamic and static pressures, battery voltage, and temperature) are read directly from the hardware via the DSC's ADC and passed through a digital low-pass filter bank to attenuate sensor noise. The accelerometers and rate gyros are queried via a SPI, and the digital stream is decoded according to the device specifications. The magnetometers are configured and queried through I²C, implemented as an interrupt-driven state machine to decode the data.

The analog sensor, accelerometer, rate gyro, and magnetometer data are passed through a calibration and temperature compensation block to produce temperature-compensated data in meaningful engineering units. The algorithms and test procedures are detailed later. The data being received by the input capture port from the radio control receiver (RCRx) are received and scaled from the PWM duty cycle to angular values in radians corresponding to control surface deflection commands. The data are assembled into a data structure that is passed to other parts of the code for further calculations.

In hardware-in-the-loop (HIL) simulation (see Part 1) and in real flight, the attitude and position estimation algorithm (shown in green) processes the data and sends the results to the control DSC via a communications protocol encoder (which separates the data into sentences and manages the SPI inter-processor communications bus).

The control DSC's low-level software architecture (shown in Figure 2) is currently less resource intensive than that of the sensor DSC. The packets sent from the sensor DSC are received and placed in a circular queue. The inter-processor communications (IPC) protocol decoder reads from the queue and assembles the data for the navigation and control algorithm (shown in green).

In a similar manner, data received from the radio modem—containing ground control station (GCS) commands—are stored in a separate circular queue. The GCS commands protocol decoder reads the queue, interprets those commands, and passes them on to the navigation and control algorithm. The data being received by the RCRx are scaled from the PWM duty cycle to angular values and passed to the guidance and control algorithm to be used as initial conditions for the manual-to-autonomous transition. This ensures that no sudden transients are experienced by the aircraft when switching between manual and autonomous mode. In other words, an instant after telling the autopilot to fly the airplane, the control surfaces should be in exactly the same place that they were before the switch.

The guidance and control algorithm generates the required control surface commands, which are scaled from angular values to the correct PWM duty cycle and sent to each of the control surface's servos. These commands are also passed on (along with the data sent from the sensor DSC) to a protocol encoder, which splits them into different sentences, schedules them for transmission, and sends them to the radio modem for ground telemetry data. These data are used to drive the GCS display, recorded for post-flight analysis, and used for both control loop tuning and fault diagnosis in real time.

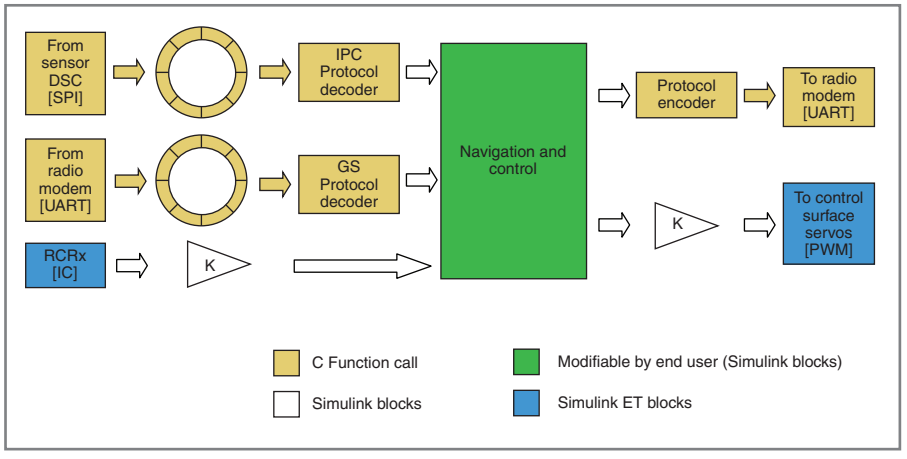

Figure 2—Low-level software architecture for the control DSC

COMMUNICATIONS PROTOCOL

The complete autopilot system generates 168 different meaningful variables. Attitude, position, sensor biases, temperature, absolute and differential pressure, control surface commands, controller gains, and many more are generated on every cycle. These values are not only used by the autopilot, but also sent to the GCS and logged to facilitate debugging and post-flight analysis. During our initial development phase, we produced a very simple, yet effective, binary communications protocol, where each message contained a 4-byte header and a 3-byte trailer. Nevertheless, we soon realized that adding a new sentence to the protocol was tedious and error-prone. After a detailed evaluation of possible solutions, we decided to adopt the

open-source communications protocol MAVLink developed by Lorenz Meier at ETH Zurich. This protocol is a lightweight, message-marshalling library for autonomous vehicles that allows the serialization of data structures (C-structs in particular) for transmission. But the feature that tipped the scales toward selecting MAVLink as the communications protocol of choice was the capability to automatically generate all of the required C code to serialize, pack, send, receive, and decode these data structures. One only needs to describe the fields of the packets at a high level using a simple XML syntax, and the tools available with the protocol will generate all of the required C code. Furthermore, the project itself has great documentation on how to integrate MAVLink into an existing project (provided the project is coded in C).

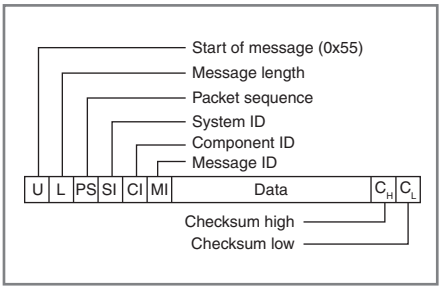

Figure 3—Communications protocol message structure

Figure 3 shows how the protocol's packets are structured. There is a 1-byte packet start sign (the letter U). Payload data specifies the size of the actual data being transmitted. A packet sequence is a counter that gets incremented as packets are sent out. The system and component identifiers uniquely identify the source of the message. A message identifier indicates what type of message it is. After the payload data, a 2-byte checksum is appended to the message to verify its validity upon reception.

Due to limitations in the RF link bandwidth, some of these messages are sent at a high data rate (100 Hz), while others that do not change very often are sent at a lower data rate (10 Hz). The protocol implements a small set of "spare" messages for debugging that can be assigned to any variable or calculation accessible within the autopilot software, both for floating-point and integer values. These are automatically logged and displayed on the GCS, and have proven very useful when testing out new algorithms.

TEMPERATURE & SENSOR COMPENSATION

The sensors used in the SLUGS are relatively inexpensive MEMS devices. As a result, they tend to have larger noise, nonlinearities, and temperature drifts than traditional (and more expensive) inertial sensors. The largest error seen in these low-cost sensors is a null-point shift as a function of temperature. Temperature compensation is used on the SLUGS to attenuate this null shift via an onboard, low-power linear Microchip Technology MCP9701 active thermistor. This is inexpensive and requires very little board real estate.

To determine which sensors within the suite required temperature compensation, we conducted the following experiment. The board was sealed inside a Ziploc bag and placed inside the lab's freezer. After an hour, the board was taken out, powered, started data recording, and placed on top of an electric grill to generate a slow temperature sweep (see Photo 1).

In order to find the temperature dependence for a given sensor, using the captured data, we performed a scatter plot of temperature versus sensor readings. From that plot, a simple linear coefficient (delta sensor counts/delta temperature counts) was extracted using a least squares fit. The sensor reading is then adjusted using this coefficient. The new sensor reading is the raw plus the coefficient multiplied by the difference in temperature from a nominal point.

The advantage is that this dramatically reduces the temperature drift of the sensors; the disadvantage is that the sensor reading is now a function of both the sensor and the temperature sensor, possibly increasing the noise in the calibrated sensor output.

Lastly, a more careful temperature curve could be used instead of the simple linear correction, although this was deemed effective for our needs. Figure 4 shows the importance of temperature compensation. It shows the temperature run (top left) and compares it with the readings from the static pressure sensor (top center). A clear correlation exists between the two. As expected, the scatter plot (top right) confirms the dependence between the two and shows the linear temperature coefficient. The resulting reading after temperature compensation (bottom plot, red) shows a significant improvement over the uncompensated reading (bottom plot, blue). Although temperature compensation does not completely remove the sensor's temperature dependence, it corrects the majority of the effect.

Bias points on the sensors are relatively easy to calibrate using static data (and several are estimated on-the-fly by the attitude and position estimation algorithms). In order

Photo 1—The SLUGS autopilot on top of an electric grill used to record data for temperature compensation calculations

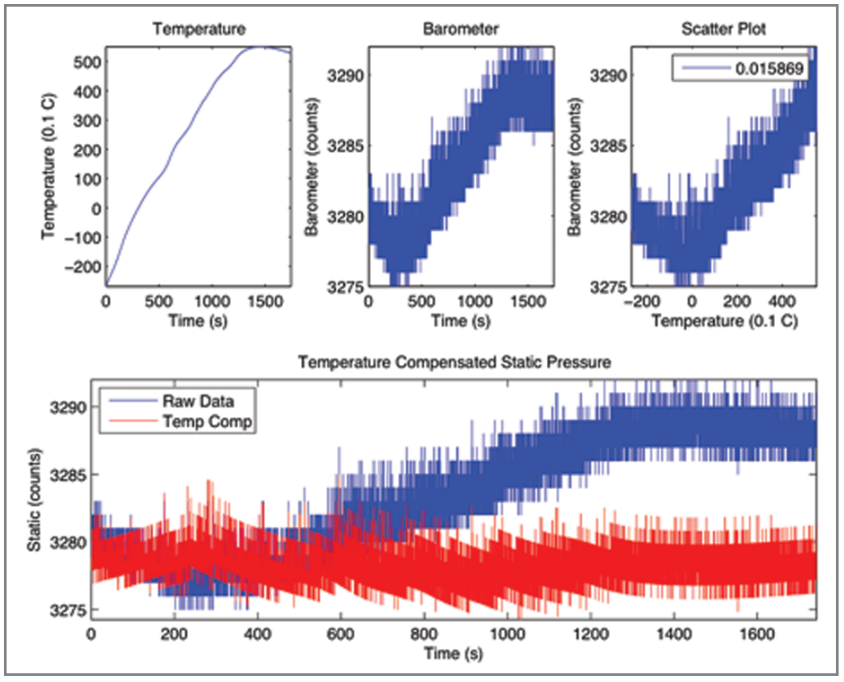

Figure 4—Static pressure sensor (barometer) temperature compensation results

to match scale factors, two different methods were used depending on the sensors. For the gyros, they were matched against a better sensor, a KVH DSP-3000 fiber optic gyro (FOG). A one-axis rate table and a least squares process were used to find the best scale factor match between the MEMS gyros and the FOG.

For the accelerometers and magnetometers, an ellipse method was used to simultaneously find scale factor, null shift, and non-orthogonality errors. The three axis sensors measure the x, y, and z components of a single (nonzero) vector. If the sensors were ideal, and you rotated the body around all axes and plotted the x, y, and z measurements, you would see points on the surface of a sphere. When plotting the real measurement instead of a sphere, you get a rotated ellipse. The center of the ellipse corresponds to the null shift, the semi-major axes to the scale factors, and the rotation to non-orthogonality within the axes. In the case of the magnetometer, these are often referred to as hard- and soft-iron effects. In practice, the aircraft was tumbled out at the field while telemetry data was stored on the ground station in order to get as close to the flying configuration as possible. (The specific algorithms that back out the parameters to correct the accelerometers and magnetometers can be found in the References section of this article.) Photo 2 shows the aircraft being tumbled by one of us prior to a test flight. (This is necessary to do only once, not before every flight.)

Lastly, the UAV was carefully leveled into flying attitude in the lab using a high-accuracy spirit level. The autopilot was turned on and allowed to converge on its attitude. This was used to generate a small rotation offset matrix to correct for the difference between the autopilot mounting and the aircraft body coordinates. This is held as a direction cosine matrix (DCM) in order to correctly account for the rotation. (You cannot simply add angles, and this does not preserve the mathematical relationships for rotation sequences.)

POSITION & ATTITUDE ESTIMATION

The initial design of the attitude filter used a complementary filter with a magnetometer triad for a heading reference and estimates of inertial velocity from the position filter. This design was placed in a Simulink simulation of 6DOF UAV flying a closed course defined by waypoints requiring changes in altitude and track. The original design did not include longitudinal acceleration (dV/dt in body coordinates), but this was found to be necessary to account for changes in speed. Several candidates for computing this term were compared on the basis of tracking errors before choosing a high-pass filter to compute the approximate numerical derivative.

Two attitude filters were designed. One used magnetometers for a heading reference. The other used GPS COG. The COG design was created because of the possibility of the high noise magnetometer readings (which were, in fact, experienced in the field) due to the UAV's electric motor.

The initial design for the position filter was a nine-state extended Kalman filter (EKF) with state variables consisting

Photo 2—Aircraft tumbling while recording data for sensor calibration

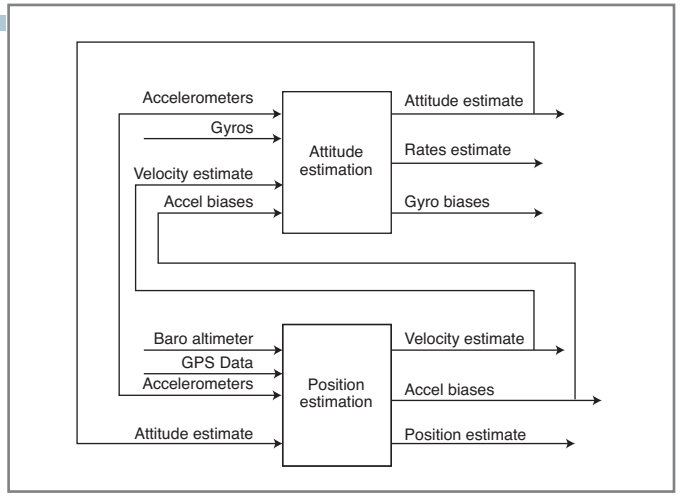

Figure 5—Position and attitude filter architecture

of position and velocity in North-East-Down (NED) coordinates and accelerometer biases. The input measurements were horizontal GPS position and velocity, barometric altitude, accelerometer readings, and the direction cosine matrix from the attitude filter. The filter performed strapdown navigation calculations for the state and covariance matrix at 100 Hz between GPS updates by integrating accelerometer readings after they were transformed to NED coordinates. This filter performed well in the six DOF simulations with noise characteristics taken from the sensor specifications. It was downloaded to the autopilot for further testing.

The objective of the first few flights was to log sensor readings while the UAV was manually controlled to validate attitude and position estimates before using these outputs for closed-loop guidance and control. Magnetometer readings were taken in the field during static ground testing before the first flight, and they were deemed unreliable. Thus, the attitude filter with COG heading reference was downloaded to the autopilot in the field and used for the remainder of the flights.

Sensor readings were logged during manually controlled flight. These data were then replayed after the flight through the Simulink position and attitude models. Unfortunately, the accelerometers' in-flight readings contained a high level of noise and the output of the EKF was unusable. These telemetry data also showed that the GPS readings had little noise (except for altitude drift and very large jumps in altitude readings) and GPS updates were averaging 3 Hz.

Due to the relatively clean GPS data and the fast GPS update rate, we replaced the EKF with complementary filters for horizontal position and velocity. For the vertical channel, barometric altitude was blended with GPS altitude readings to eliminate drift. The outlier changes in altitude readings were trapped with a threshold test before further processing.

The new filter design reduced the computational load on the sensor DSC from near 100% to 40%. Using the SLUGS environment, the elapsed time—from beginning the redesign to the first download to the autopilot—was less than 24 hours, demonstrating the flexibility and usability of the SLUGS for research purposes.

The final design performed extremely well, as shown by the excellent performance of the UAV under autonomous control. In addition, the attitude and position filters did extremely well when compared with more sophisticated models using other flight-test data (see Figure 5).

NAVIGATION & CONTROL

The control system designed for the SLUGS is based on the successive loop closure (or inner/outer loop) approach for aircraft control. It is completely implemented in the control DSC (see Part 1 of this series). The inner loop comprises a set of proportional integral derivative (PID) controllers separated into lateral (pitch and airspeed) and longitudinal (roll and yaw) channels. These are designed to stabilize the UAV and receive mid-level commands of turn rate, altitude, and airspeed. This loop generates the direct commands to the control surfaces: ailerons, rudder, elevator, and throttle. These each contain saturation limits on the servos, and can be individually tuned in flight. The outer loop implements a high-level, waypoint-based navigation, which in turn generates commanded turn rate and altitude for the inner loop. Airspeed is set to a constant and held by the inner loop PID controller. Altitude is controlled with two cascaded PID loops, one to command pitch based on altitude error with a strong saturation limit (set from the GCS to 15°), and a second to command the elevator to achieve the desired pitch output from the first PID stage.

The autopilot itself has three different submodes of operation. The first mode is Pilot Feedthrough, where the pilot commands are received by the RCRx and passed directly through the autopilot untouched.

The second mode is Manual Direct, where the GCS sends mid-level commands directly to the autopilot. Manual direct can be combined with Pilot Feedthrough such that

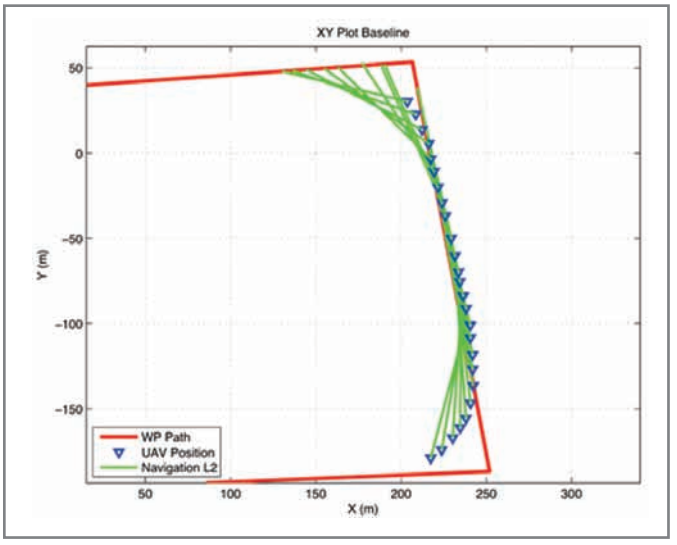

Figure 6—Time progression of the UAV position and the computation of the L2 vector during a flight test

the safety pilot controls some of the control surfaces, but the autopilot others. This is extremely useful when tuning control gains. For instance, throttle is given to the autopilot, but all other controls remain with the safety pilot. The airspeed control loop is tuned for decent behavior while the safety pilot can keep the UAV within the bounds of the test area. Furthermore, the safety pilot can induce errors by climbing or diving the UAV and seeing if the throttle control responds correctly.

The third submode is Autonomous-waypoint Navigation, in which the GCS operator configures the waypoints, and the navigation algorithm becomes responsible for generating the mid-level commands. All modes receive configuration settings from the GCS operator. The inner loop gets the PID gains for each controller. The outer loop receives the waypoints (in latitude, longitude, and height) and additional configurations that are necessary for its operation.

In our project, the UAV is tracking a point that is on the path from the previous waypoint to the next one that is ahead of the UAV. The "look ahead" length, called "L2," is scaled by ground velocity, making the aircraft less sensitive in a tailwind and more sensitive to cross-track errors in a headwind. The angle between the aircraft and the L2 point is used to command the lateral acceleration of the aircraft (conveniently, this corresponds to a command-in-roll angle for the UAV). Tighter control is achieved by shortening the look-ahead distance, but at the risk of oscillations about the path. Figure 6 shows data from an actual flight path of the UAV in the presence of wind. The L2 vector is shown in green, the UAV's position is in blue, and the commanded path is in red.

FLIGHT TEST RESULTS

The UAV employed for flight tests was a modified version of an off-the-shelf Multiplex Mentor radio control (RC) aircraft. Modifying the aircraft's fuselage involved shaving out a significant section of the "cockpit"' (see Photo 3a) to make way for a wireframe pod (see Photo 3b). This pod, made out of balsa wood, securely housed the autopilot and the radio modem (see Photo 3c) in the airframe. The pod has two functions. One, it protects the main avionics in case of a crash. And two, it provides a vibration-isolated environment on which to mount the autopilot. This also makes the core avionics trivial to mount and unmount from the UAV for lab testing and HIL simulation.

Flight tests were performed at UCSC's east field. During the first three weeks of flight tests, the UAV was flown under pilot control. The collected flight data were analyzed offline to verify and validate the inner workings of the autopilot. This initial test period was also used to tune the aforementioned position and attitude estimation filters. Once the position and attitude estimation filters were proven to work, one week of flight tests (more than nine flights) was devoted to tune the PID gains for the inner loop. The tuning process

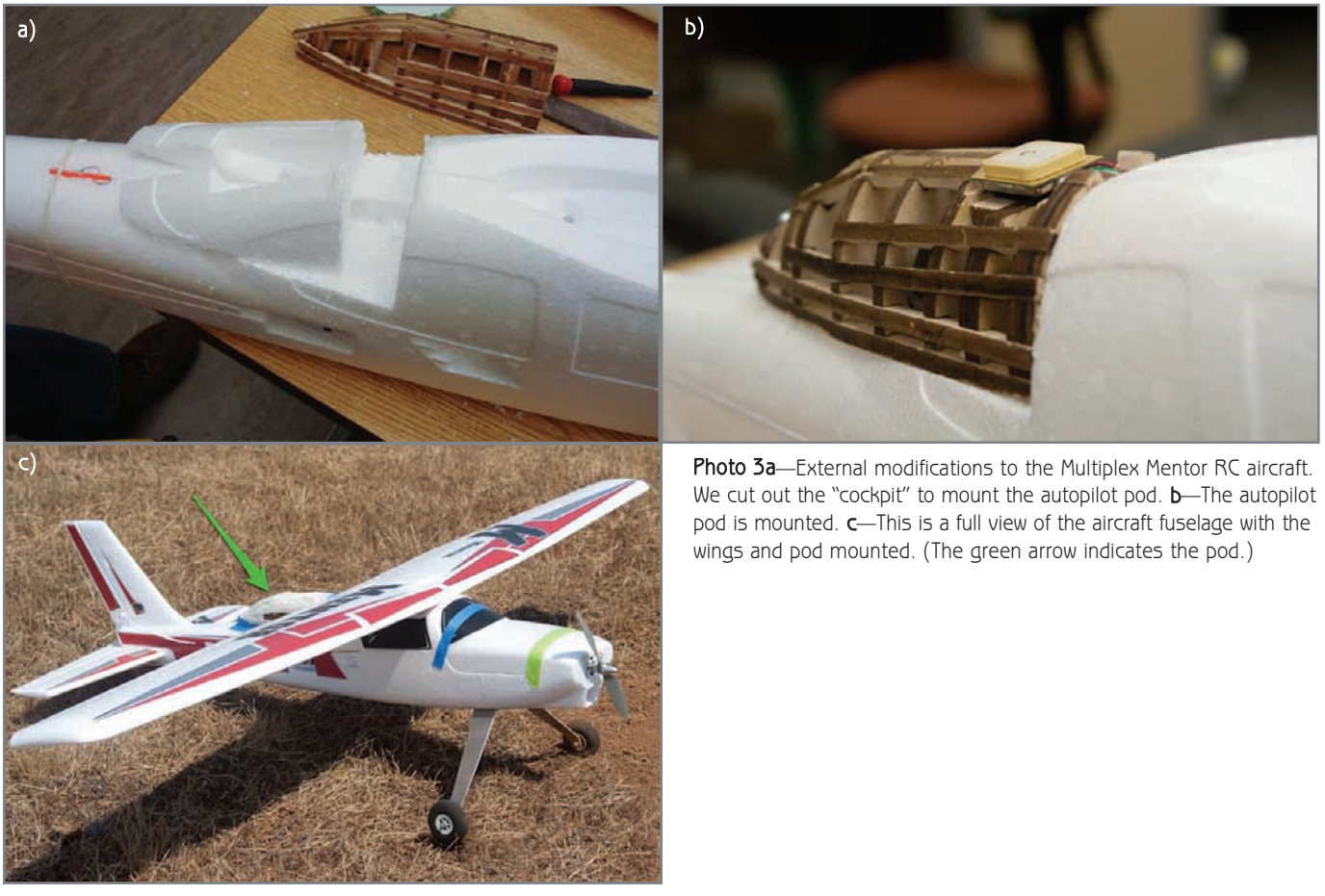

Photo 3a—External modifications to the Multiplex Mentor RC aircraft. We cut out the "cockpit" to mount the autopilot pod. **b**—The autopilot pod is mounted. **c**—This is a full view of the aircraft fuselage with the wings and pod mounted. (The green arrow indicates the pod.)

involved adjusting the gain of each controller one at a time during flight while the safety pilot retained partial control of the aircraft. Any time it was deemed necessary, the safety pilot could recover complete control by flipping a switch on his transmitter that turned the analog MUX back to pass through. Eventually, the safety pilot only had control of the rudder, using it to skid the UAV through turns to keep it within the test area while the inner loop stabilized airspeed, roll, altitude, and turn rate.

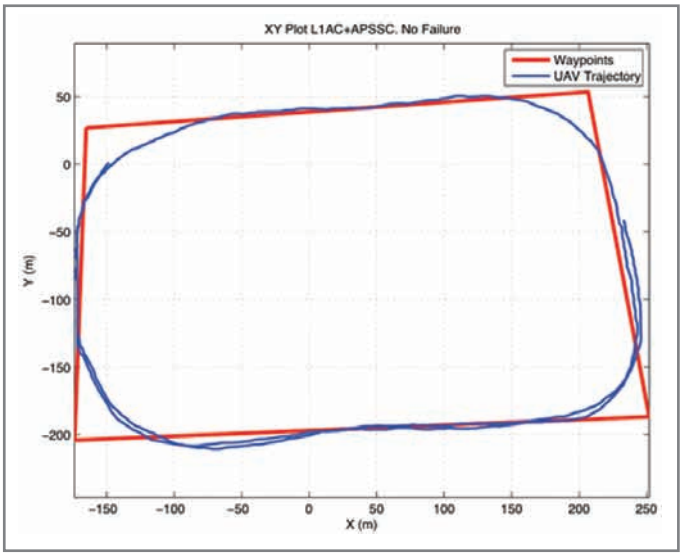

Figure 7—Trajectory plot of the UAV under autonomous control. The ground station was physically located at the origin.

The primary objective of experimental testing was to validate the results correspondence between Simulink and HIL simulations and flight tests. The secondary objective was to validate the complete workflow in the SLUGS: Simulink to HIL simulation to flight test. The waypoint track set for the flight test experiments consisted of four waypoints defining two short and two long legs (roughly defining a rectangle). The flights were always performed within visual range of the safety pilot, thus the short separation between waypoints. For every flight test, the UAV took off under pilot control. The pilot would do several circuits around the path to make sure everything was working correctly, then release the control to the SLUGS autopilot. Figure 7 shows a typical route around the preprogrammed waypoint track under autonomous control.

It is important to stress that all of the SLUGS's control and navigation loops were implemented directly in Simulink. A high-level programming language (such as C, C++, or Java) wasn't used. The Simulink models used for software simulation, HIL simulation, and flight tests are exactly the same. These were first used in Simulink and then compiled and downloaded to the autopilot hardware. The navigation loop was shown to perform robustly, with good tracking results even in the presence of turbulence and wind. The inner and outer loops performed well, and we achieved basic flight and waypoint following.

TAKING FLIGHT

We presented a top-level view of the SLUGS, an open-source UAV autopilot, its ground control station, and its HIL simulator. We presented results for its sensors, temperature compensation, and calibration. Using Simulink as the primary development platform, UAV researchers can easily transition between software simulation, HIL simulation, and flight testing without the need to re-code the algorithms in a traditional programming language.

The SLUGS was flight tested under nominal flying conditions with successful results, but there is room for improvement. We recently adopted the open-source GCS software QGroundControl. This project is in active development and has a thriving community of contributors; but, at its beginnings, it was mostly focused on autonomous quad-copter support. This has changed, and now there is a wide range of tools to support autonomous airplanes. The SLUGS has been slowly adopting all these features.

Also, we're taking steps to adapt the SLUGS's code base so it can work in a wider variety of autonomous vehicles. In particular, work is being done to use the SLUGS in an autonomous boat, a VTOL aircraft, and a ground vehicle. Finally, the SLUGS is currently being adopted in several educational research laboratories in the United States and elsewhere. This will undoubtedly help improve the overall quality of this project as a community of users each adds their own improvements.

Authors' note: We would like to express our deepest gratitude to Greg Horn for working so hard on the board schematics and layout and hand-soldering the components of the first two batches of autopilots. He was also our safety pilot, and more than once he saved us from having to build another UAV. We would also like to thank Vladimir Dobrokhodov for all of his help in making the SLUGS happen and lending his experience as a long-time UAV developer and end user. Finally, we would like to thank Lorenz Meier at ETH Zurich for all of his help in integrating MAVLink into the SLUGS.

Mariano Lizarraga (malife@soe.ucsc.edu) holds a BS in Naval Sciences Engineering from the Mexican Naval Academy, an MSEE and Electrical Engineer's degree from the U.S. Naval Postgraduate School, and a PhD in Computer Engineering from UCSC. He has more than eight years of experience writing software and designing hardware for UAVs. He is currently a Research Fellow in the Computer Engineering Department at the University of California Santa Cruz.

Renwick Curry (rcurry@ucsc.edu) earned an AB in Physics from Middlebury College, and he holds a BS, MS, and PhD in Aeronautics and Astronautics from MIT. He has worked as a faculty member (MIT and Cornell), a spacecraft guidance engineer (Mariner and Apollo projects), and an aviation safety engineer for NASA. Renwick is currently an Adjunct Professor in the Computer Engineering Department at University

of California Santa Cruz, and president of AASI (www.aasi.com), a company specializing in aircraft flight-path optimization.

Gabriel Hugh Elkaim (elkaim@soe.ucsc.edu) holds a BSE in Aerospace Engineering from Princeton University, as well as an MS and PhD in Aeronautics and Astronautics from Stanford University. He is an Associate Professor in the Computer Engineering Department at University of California Santa Cruz, where he works on autonomous vehicle guidance, navigation, and control.

RESOURCES

M. Euston, P. Coote, R. Mahony, J. Kim, and T. Hamel, "A Complementary Filter for Attitude Estimation of a Fixed-Wing UAV," Australian National University College of Engineering & Computer Science, 2008.

GitHub Inc., Source code repository, http://github.com/malife/SLUGS

L. Kerhuel, Simulink Embedded Target for PICs, www.kerhuel.eu/wiki/Index.php5, 2010.

R. Mahony, T. Hamel, and J. Pflimlin, "Nonlinear Complementary Filters on the Special Orthogonal Group," *IEEE Transactions on Automatic Control*, Vol. 55, No. 5, 2008.

S. Gleason and D. Gebre-egiabher, *GNSS Applications and Methods (GNSS Technology and Applications)*, Artech House Publishers, 2009.

S. Park, J. Deyst, and J. P. How, "Performance and Lyapunov Stability of Non-Linear Path Following Guidance Method," *Journal of Guidance, Control and Dynamics*, Vol. 30, 2007.

QGroundControl, MAVLink Micro Air Vehicle Communication Protocol, www.qgroundcontrol.org

SLUGS Project, http://slugsuav.soe.ucsc.edu

UC Santa Cruz Autonomous Systems Lab, http://asl.soe.ucsc.edu

SOURCES

DSP-3000 Fiber optic gyro (FOG)
KVH Industries, Inc. | www.kvh.com

dsPIC33 DSC and MCP9701 Thermistor
Microchip Technology, Inc. | www.microchip.com