JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 3, May 2006

A Compact Guidance, Navigation, and Control System for Unmanned Aerial Vehicles

Henrik B. Christophersen,* R. Wayne. Pickell,* James C. Neidhoefer,* Adrian A. Koller,† Suresh, K. Kannan,* and Eric N. Johnson‡
Georgia Institute of Technology, Atlanta, GA, 30332-0150

The Flight Control System 20 (FCS20) is a compact, self-contained Guidance, Navigation, and Control system that has recently been developed to enable advanced autonomous behavior in a wide range of Unmanned Aerial Vehicles (UAVs). The FCS20 uses a floating point Digital Signal Processor (DSP) for high level serial processing, a Field Programmable Gate Array (FPGA) for low level parallel processing, and GPS and Micro Electro Mechanical Systems (MEMS) sensors. In addition to guidance, navigation, and control functions, the FCS20 is capable of supporting advanced algorithms such as automated reasoning, artificial vision, and multi-vehicle interaction. The unique contribution of this paper is that it gives a complete overview of the FCS20 GN&C system, including computing, communications, and information aspects. Computing aspects of the FCS20 include details about the design process, hardware components, and board configurations, and specifications. Communications aspects of the FCS20 include descriptions of internal and external data flow. The information section describes the FCS20 Operating System (OS), the Support Vehicle Interface Library (SVIL) software, the navigation Extended Kalman Filter, and the neural network based adaptive controller. Finally, simulation-based results as well as actual flight test results that demonstrate the operation of the guidance, navigation, and control algorithms on a real Unmanned Aerial Vehicle (UAV) are presented.

Nomenclature

$a_{IMUx}, a_{IMUy}, a_{IMUz}$	acceleration measurement from the IMU
b_{ax}, b_{ay}, b_{az}	accelerometer measurement biases
$b_{\omega x}, b_{\omega x}, b_{\omega x}$	rate gyro biases
f	the process model
F_k	the process model Jacobian at t_k
g	the gravitational constant
h_t	altitude of ground level
h_{cg}	altitude of the center of gravity (c.g.)
h_k	the measurement model at time t_k
H_k	the measurement model Jacobian at t_k
h_{sensor}	altitude of the pressure sensor

Received 27 July 2005; revision received and accepted for publication 17 November 2005. Copyright 2005 by Henrik B. Christophersen, R. Wayne. Pickell, James C. Neidhoefer, Adrian A. Koller, Suresh, K. Kannan and Eric N. Johnson. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission. Copies of this paper may be made for personal or internal use, on condition that the copier pay the $10.00 per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923; include the code 1542-9423/04 $10.00 in correspondence with the CCC.
* Research Engineer, Aerospace Eng. Dept., Georgia Institute of Technology, Atlanta, GA, 30332-0150.
† Graduate Research Assistant, Aerospace Eng. Dept., Georgia Institute of Technology, Atlanta, GA, 30332-0150.
‡ Lockheed Martin Assistant Professor of Avionics Integration, Aerospace Eng. Dept., Georgia Institute of Technology, Atlanta, GA, 30332-0150.

K_k	Kalman Filter gain at time t_k
ω	angular velocity vector
P_k^-	predicted error covariance matrix at time t_k
P_k^+	corrected error covariance matrix at time t_k
$Q(t)$	process noise covariance matrix at time t
q_0, q_1, q_2, q_3	quaternion parameters
r_{GPS}	position vector of GPS relative to the c.g.
r_{sensor}	position vector of sensor relative to the c.g.
R_k	measurement noise covariance matrix at time t_k
Δt	integration time step
$T_{b->i}$	transformation from body frame to inertial frame
$T_{b->i}[3]$	third row of $T_{b->i}$
t_k	the time at which the k^{th} measurement is taken
u	x-body axis velocity
$u(t)$	control vector at time t
u_k	process control vector at time t_k
v_k	measurement noise at time t_k
v	y-body axis velocity
v_{cg}	velocity of the c.g.
v_{GPS}	velocity of the GPS in and inertial frame
w	z-body axis velocity
$w(t)$	process noise at time t
w_{sensor}	pressure sensor measurement noise
w_x	GPS position measurement noise
w_v	GPS velocity measurement noise
w_ω	IMU angular rate measurement noise
w_a	IMU acceleration measurement noise
$x(t)$	process model state
x_{pos}	x position
x_{posGPS}	position of the GPS in an inertial frame
x_{poscg}	position of the c.g.
x_k	process state at time t_k
$\hat{x}_k^-$	predicted state estimate at time t_k
$\hat{x}_k^+$	corrected state estimate at time t_k
y_{pos}	y position
y_k	output estimate at time t_k
z_k	observations at time t_k
z_{pos}	z position

List of Acronyms

A/D	Analog to Digital
ADC	Analog to Digital Converter
BGA	Ball Grid Array
COTS	Commercial-Off-The-Shelf
CPU	Central Processing Unit
D/A	Digital to Analog
DAC	Digital to Analog Converter
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
EC20	FC20 Processor Board

EDMA	Enhanced Direct Memory Access
EKF	Extended Kalman Filter
EMIF	External Memory Interface
FIFO	First-In First-Out
FLOPS	Floating Point Operations Per Second
FCS20	Flight Control System Version 20
FPGA	Field Programmable Gate Array
GN&C	Guidance, Navigation, and Control
GPS	Global Positioning System
GUI	Graphical User Interface
IC	Integrated Circuit
IMU	Inertial Measurement Unit
I/O, IO	Input/Output
JTAG	Joint Test Access Group
LC	Logic Cell
MEMS	Micro Electro-Mechanical System
MIPS	Millions of Instructions Per Second
NAC	Neural Adaptive Controller
NED	North, East, Down
OS	Operating System
PCB	Printed Circuit Board
PD	Proportional, Derivative
PDA	Personal Digital Assistant
PLL	Phase Locked Loop
PWM	Pulse Width Modulated
ROM	Read Only Memory
RTOS	Real Time Operating System
SB20	FCS20 Sensor/Power Board
SDRAM	Synchronous Dynamic Random Access Memory
SVIL	Support Vehicle Interface Library
UART	Universal Asynchronous Receiver Transmitter
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
UAV	Unmanned Aerial Vehicle

I. Introduction

AS the capabilities of Unmanned Aerial Vehicles (UAVs) expand, increasing demands are being placed on the hardware and software that comprise their Guidance, Navigation, and Control (GN&C) systems. In addition to standard functions such as data acquisition, filtering, stability augmentation, and tracking control, state-of-the-art GN&C systems must now support more advanced functions such as automated decision making, obstacle avoidance, target acquisition, target tracking, artificial vision, and interaction with other manned and unmanned systems. Furthermore, interest in small, mini, and micro UAVs is growing. So, while GN&C system performance requirements are increasing, the acceptable form factors (size and weight) of these systems are decreasing.

A small, integrated GN&C hardware and software system, referred to as the Flight Control System Version 20 (FCS20), has recently been developed to maximize this ratio of performance to size, with the capability of supporting advanced algorithms in a very compact package. The FCS20 uses both Field Programmable Gate Array (FPGA) and Digital Signal Processor (DSP) technology to enable custom vehicle interfacing and fast sequential processing of high-level GN&C algorithms.

This miniature computer uses its floating point Digital Signal Processor (DSP) for high level serial processing, its Field Programmable Gate Array (FPGA) for low level parallel processing, and Micro Electro Mechanical Systems

(MEMS) sensors. In addition to controlling the vehicle, the FCS20 can also support payload control, image processing, communications interfaces (encryption), vehicle health monitoring, and other high level algorithms. Using a single FCS20 to support these systems differs from the more traditional approach of using separate and dedicated hardware components.

The maximization of the performance-to-size ratio in the FCS20 was achieved largely during the design process by creating a "purebred" system for GN&C type applications. Many modern flight control systems are rendered bulkier and more inefficient by hardware, operating systems, and low level software that contain a great deal of overhead functionality, such as video and sound outputs, keyboard data entry, GUIs, etc. The FCS20, on the other hand, uses very streamlined and specialized hardware and software systems that can support high levels of performance with relatively low power and space requirements. Furthermore, the FCS20 still retains much of the flexibility and functionality of larger systems due to its expandability and programmability.

To support customization for a wide range of applications, the FCS20 uses a modular building block strategy to link its various components. The basic modules of the FCS20 are the EC20 processor board and the SB20 sensor/power board. A minimal FCS20 system consists of one EC20 board and one SB20 board, usually set up with the boards connected back to back. Expanded bank or stack configurations with multiple EC20 boards or other application-specific boards can be assembled in various physical arrangements.

While the FCS20 may be slightly more expensive (due to its FPGA) than other small, commercially available, stand-alone GN&C hardware/software systems, it offers advantages in terms of both size and performance. Some commercially available systems include the Piccolo by Cloudcap Technolgy,[1] the MP2028 g by Micropilot,[2] the AP50 by UAV Flight Systems, Inc.,[3] and the GS111 m by Athena Controls.[4] These systems all use similar sensor suites as well as processors for data acquisition, filtering, and control and I/O for interfacing with servos, data links, etc. They also all offer multiple control modes, such as airspeed and altitude control and 3D way-point navigation. All of these systems include ground support software and are designed to operate with a wide variety of UAV platforms. In terms of size, only the Micropilot MP2028 g is smaller than the FCS20, with a 100×40 mm board plan-form versus an 85×55 mm FCS20 board. In terms of sensing performance, the capabilities of all five systems are comparable. They all incorporate inertial rate gyros and accelerometers, altitude sensors, and integrated GPS, as well as some type of filter to estimate attitude. However, of the five systems, only the FCS20 offers the computing flexibility and performance of both a DSP and an FPGA, and is capable of supporting advanced functions such as automated decision making, obstacle avoidance, target acquisition, artificial vision, etc.

The FCS20 can also be compared to several COTS products that contain both DSPs and FPGAs on the same board. These include the ADDS-21261/Cyclone Evaluation Platform,[5] the Micro-Line C6713Compact,[6] and the SignalWave.[7] The ADDS-21261/Cyclone is produced by Danville Signal, Analog Devices, Altera, and Arrow Electronics. This system combines an Analog Devices ADSP-21261 DSP and an Altera EP1C3 Cyclone FPGA on a single board. The Micro-Line C6713Compact is produced by Traquair. It incorporates a high performance TMS320C6713 floating-point DSP Processor, a Virtex-II FPGA, an IEEE 1394 FireWire communications interface, up to 64MBytes of SDRAM, and 8MBytes of FLASH ROM. The SignalWave is produced by Lyrtech. It uses a Texas Instruments TMS320C6713TM DSP, a 3M gates Xilinx Virtex-IITM FPGA, and high speed ADC and DAC up to 65 MHz. While the SignalWave is intended for use with audio systems, the Cyclone and the Micro-Line are both general use systems. All of these systems are capable of interfacing with a wide variety of peripheral devices; however, none has a dedicated sensor board suitable for autonomous UAV operations, nor are the authors aware of any of these systems having been used for UAV operations.

This paper focuses first on three major aspects of the FCS20: computing, communications, and information. The computing aspects of the FCS20 include details about the design process, hardware components, board configurations, and specifications for EC20 Processor board and the SB20 Sensor/Power board. The communications aspects of the FCS20 include descriptions of internal and external data flow. The information section describes the FCS20 Operating System (OS), the Support Vehicle Interface Library (SVIL), the Navigation Kalman filter, and the neural network based adaptive controller. Finally, flight test results of the FCS20 controlling an 11-inch ducted fan are presented. The result is a clear illustration that future small UAV systems can potentially take advantage of far more complex onboard processing than is the case today. Along the way, several specific methods are described; methods that are part of one approach to achieving this objective.

II. *Computing:* The FCS20 Hardware

A. The EC20 Processor Board

1. EC20 Background

The EC20 Processor board handles FCS20 processing and internal and external communications. Early design approaches for the FCS20 utilized COTS components, such as cell phones and Personal Digital Assistants (PDAs), that already included many of the necessary hardware features, but these designs did not achieve desired performance levels and were thus discarded. Other preliminary designs involved systems based solely on FPGAs. However, these pure-FPGA systems required that a floating-point core be implemented in the FPGA, which would have resulted in a loss of flexibility and cost efficiency.

To illustrate, a typical FPGA chip with 11,000 Logic Cells (LCs) costs approximately $220.00 or $0.02 per LC. A 40 Mflop floating point core processor (VHDL module) implemented in the FPGA would use 5000 LCs or roughly $100.00 worth of the LCs on the FPGA. On the other hand, a much more powerful 1.35 Gflop floating point DSP costs only about $35.00 and would not occupy any of the resources on the FPGA. Thus, the decision to incorporate a DSP onto the same board as the FPGA ultimately promised the best combination of high performance and low cost. The use of floating point DSP processing also simplified the programming of many of the guidance, navigation, and control functions.

Another important design decision stemmed from the desire to avoid transferring high-speed signals between boards. Thus, all high-speed components were placed on the EC20, while the lower speed components were relegated to the SB20.

This solution, in which a fast sequential processing core in the DSP is supported by a parallel system of lower-level support components in the FPGA, was implemented with a Texas Instruments 225 MHz Floating Point TMS320C6713 DSP along with an Altera 1M gate Stratix FPGA. In addition, the EC20 includes a high-speed internal data bus and few external interrupts, allowing the DSP to spend more time on high speed processing and less time on data transfers.

2. EC20 Hardware Components

The EC20 processor board has seven main components: the DSP, the FPGA, the AMD flash memory, two Micron SDRAMs, a Pericon clock driver, and a power supervisor. The Texas Instruments TMS320C6713 DSP and 780 pin ALTERA Stratix EP1S10 FPGA were chosen based on their capabilities and footprints, as well as previous experience by the authors. The 3 memory blocks, the clock driver, and the power supervisor were chosen mainly for their compact Ball Grade Array (BGA) footprints.

3. EC20 Board Configuration

The left half of Fig. 1 shows the relative locations of the main hardware components and headers on the EC20 processor board. The right half of the figure shows a screenshot of the high level PCB diagram of the board. On the EC20, a 32 bit internal bus connects the FPGA to the DSP and acts as a conduit for information flow between the two. As can be seen in Fig. 1 (left), SDRAM #1 is connected directly to the internal bus for use by the DSP. SDRAM #2 is used solely by the FPGA. The Flash memory, which is the only nonvolatile component on the board, holds the FPGA image and DSP software.

The clock driver generates the frequencies required for use by the DSP and FPGA, and the power supervisor holds the processor in reset until all voltages are at approved levels during power up. Headers P1 through P6 are used for interfacing the EC20 with the SB20 and other external components such as sensors, actuators, communications equipment, etc.

The components were arranged on the board to minimize total board size. A priority was keeping the internal bus as short as possible while still keeping the bus-accessible memory close to the bus. As a result, the DSP and FPGA are located very close to each other, and a short 32-bit parallel bus runs directly between them with just enough space to allow nearby placement of the bus accessible SDRAM and flash memory (See Fig. 1, left). Also, keeping the I/O connectors close to the FPGA I/O pins minimizes trace lengths, and thus the headers (P1–P6 in Fig. 1, left) are located very close to the edges of the FPGA. After placing the DSP, FPGA, bus, memory, and headers, the other components were arranged to minimize total required trace lengths. Once the general layout had been determined, the whole configuration was condensed onto a board the size of a credit card. The dimensions of the resulting board are labeled in Fig. 2.

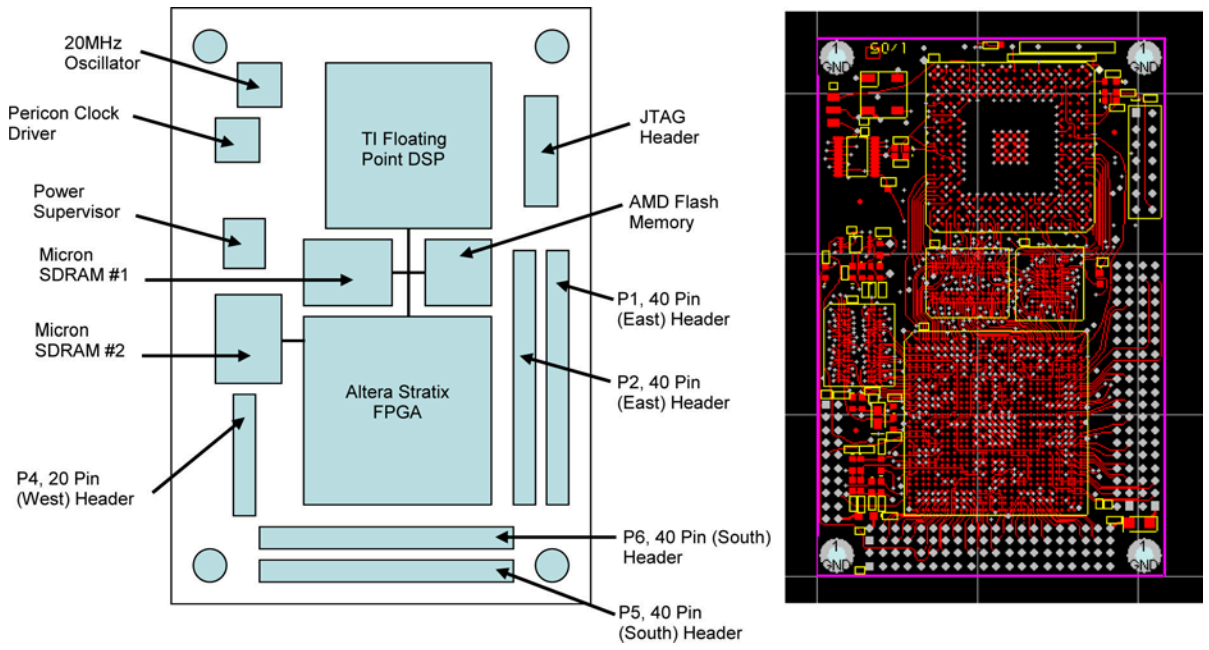

Fig. 1 The EC20 processor board configuration and high level PCB diagram.

4. EC20 Processor Board Specifications Summary

The EC20 contains all of the components required for fast data processing, including a total of 32Mbytes of SDRAM and 8Mbytes of configuration FLASH memory. Other items include power saving circuitry, 110 general purpose IO-pins, optional remote configuration control and dedicated board-to-board fast differential serial lines that operate at up to 840Mb/s. EC20 specifications are given below.

1) Texas Instrument TMS320C6713 Floating Point DSP
    a. Eight 32-bit instructions / cycle
    b. 225 MHz, 4.4 ns Instruction Cycle Time
    c. 1800/1350 MIPS/FLOPS
    d. 8K L1 cache, 256K L2 cache
    e. External Memory Interface (EMIF)
    f. Enhanced Direct Memory Access (EDMA)
    g. 272-pin BGA package
2) Altera EP1S40 Stratix FPGA (−10, −20, −25 and −30 are optional parts)
    a. 41,250 Logic Elements
    b. 3.4M internal RAM bits
    c. 14 DSP Blocks (250 MHz MACs)
    d. 112 Embedded Multipliers
    e. 12 PLLs
    f. 615 IO pins
    g. 80 high-speed differential channels optimized to 840Mbps each
    h. Support for multiple high-speed networking and communications bus standards
    i. Support for multiple Altera MegaCore functions including Nios^{TM} soft-core embedded processor
3) 128Mb (4Mx32) Micron 140 MHz SDRAM for DSP processor
4) 128Mb (4Mx32) Micron 140 MHz SDRAM for FPGA (Nios^{TM} CPU)
5) 64Mb (8Mx8) AMD FLASH memory for configuration data
    a. Eight banks internally or externally selectable

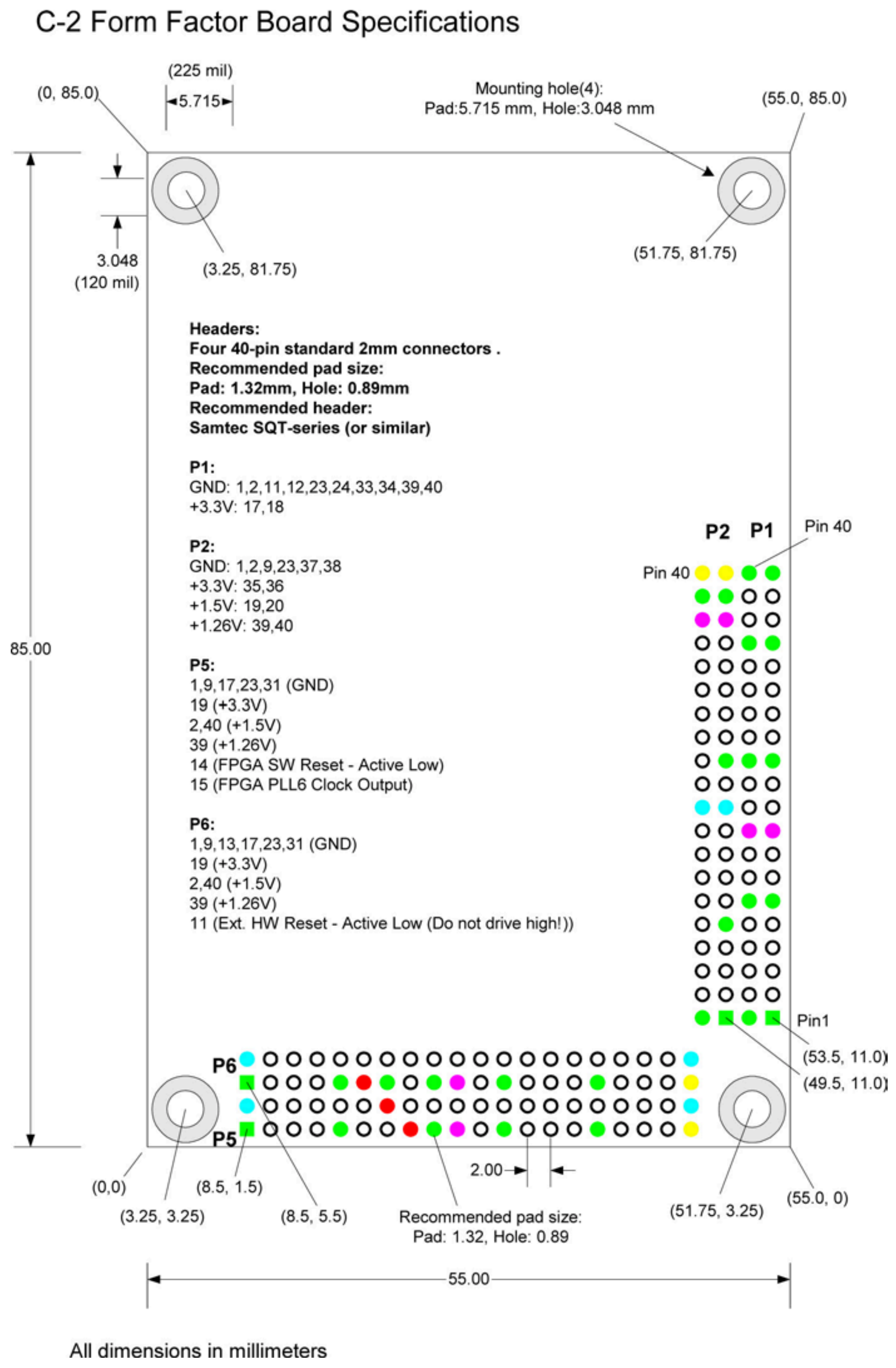

Fig. 2 The EC20 processor board form.

6) 100 MHz × 32 bit data bus (DSP, SDRAM, FPGA)
7) 100 General purpose 3.3V IO pins in three 2 mm headers
8) 10 high-speed differential channels (840Mbps each) in separate header for twisted-pair board-to-board communication networks (i.e. back plane bus)

9) Optional external Flash Bank select signals for externally controlled system configuration
10) Advanced Clock Architecture
    a. 20 MHz reference Oscillator with Zero-Skew Clock Driver
    b. Hardware configurable clock multiplier for data bus
    c. Software configurable DSP core frequency (on-the-fly)
    d. PLLs in FPGA enable multiple internal and external clock schemes
    e. Both reference clocks and PLL output clocks available at IO headers
11) Power saving features
    a. Software controlled DSP core frequency
    b. Software controlled PLLs in FPGA
    c. External reset/board shutdown
12) Power management
    a. 3.3 V, 1.5 V and 1.26 V (from power supply board)
    b. Triple voltage supervisor
13) Size: 55 mm $\times$ 85 mm
14) Weight: about 30 g

B. The SB20 Sensor/Power board

1. SB20 Background

The SB20 sensor/power board was designed to be compatible with the EC20 processor board and provides three main functions: supplying regulated and filtered power to the system, supporting onboard or external navigation sensors, and serving as an interface to external components.

2. SB20 Hardware Components

The eight main sensor components of the SB20 consist of three Analog Devices ADXR300 rate gyros, two Analog Devices ADXL210E 2-axis 10 g accelerometers, a μBlox GPS module, and Freescale absolute and differential pressure sensors. The board also contains four voltage regulators and level converters for the input/output pins used to interface with servos/actuators and other components.

3. SB20 Board Configuration

The left half of Fig. 3 shows the relative locations of the main hardware components and headers on the SB20 sensor/power board. The right half of the figure shows a screenshot of the high level PCB diagram of the board.

The SB20 uses the same form factor as the EC20 (see Fig. 2), with a layout such that the main headers line up when the boards are assembled back-to-back. In general, the layout of the SB20 was driven by the desire to locate the high frequency switching power regulators as far as possible from the sensitive circuitry in the analog sensors and A/D converters. As seen in Fig. 3, the power regulators are located on the top of the board, the digital circuitry, including the RS-232 drivers and the digital section of the GPS module, are located in the middle of the board, and the analog sensors and A/D converters are located at the bottom of the board.

In order to sense accelerations and angular rates in three axes, two sub-boards were mounted perpendicular to the main board and to each other (see Fig. 3). Each sub-board has its own ADS8341 16 bit A/D converter. These A/D converters are located immediately opposite the sub-board mounted sensors. The analog signals from the accelerometers are routed to the A/D converters through second order low pass filters, and the PWM signals from the accelerometers are routed directly to the FPGA. The analog signal from the absolute pressure sensor is routed to an AD7708 16 bit A/D converter located on sub-board #2. This A/D converter has software controlled reference voltages to enable maximum resolution. The sensors were placed to minimize the thickness of the installed board. Consideration was given to the orientation of connectors and pressure transducer ports such that ribbon cables and pressure plumbing could be routed close to the surface of the board.

In addition, the rate gyros used in the SB20 have temperature-sensing capabilities that can be used to detect the installed operating conditions inside the enclosure. The SB20 also has an A/D channel dedicated for sensing the main power supply voltage.

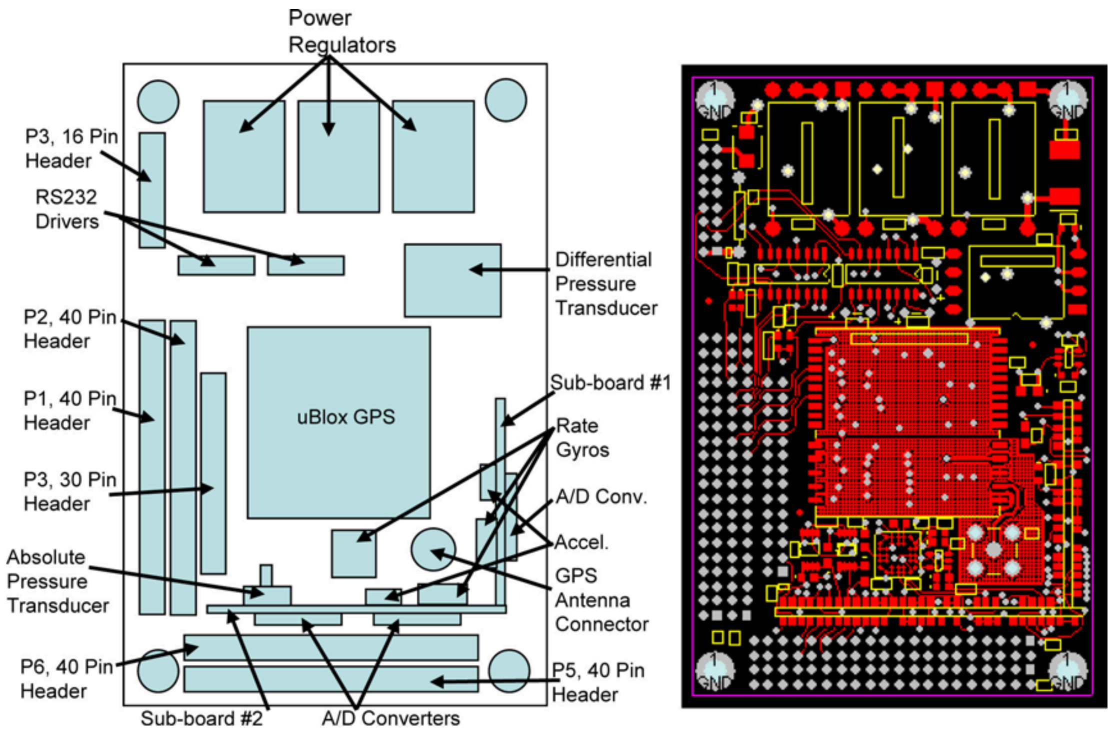

Fig. 3 The SB20 sensor/power board configuration and high level PCB diagram.

4. *SB20 Sensor/Power Board Specifications Summary*

In addition to hosting the sensor suite and distributing power to itself, the EC20, and other FCS20 components, the SB20 has four additional single ended A/D channels located on Header P4. P4 can also accept hardware and software reset signals that come from external push buttons or switches.

Highlights of the SB20 specifications are given below.

1) Three-axis IMU, Analog Devices +/−300 degrees Solid State Rate Gyros with a Temperature Sensor
2) Two 2-axis +/−10 g accelerometers (one axis redundant)
3) μBlox TIM-LF GPS module
4) FreeScale 0–15PSI Absolute Pressure Sensor (Static pressure)
5) FreeScale Differential Pressure Sensor (Dynamic pressure)
6) Two 16-bit, ADS8341 ADCs with SPI interface
7) One AD7708 8ch Sigma-Delta ADC
8) Four RS232 ports
9) 12 General Purpose 5 V IO pins
10) Power regulators providing 5 V, 3.3 V, 1.8 V and 1.26 V
11) Size: 55 mm × 85 mm × 20 mm (including GPS receiver)
12) Weight: approximately 40 g

C. FCS20 Circuit Diagrams and PCB Layouts

After selecting components and determining the general layouts for the EC20 and SB20 boards, detailed circuit diagrams and PCB layouts were developed.

The high component density of the EC20 and the SB20 necessitated careful planning of component placement, pin assignments, and manual routing. Screenshots showing the highest levels of the EC20 and SB20 PCB layouts are shown in the right halves of Fig. 1 and Fig. 3 respectively. Finally, after the routing was completed, pin designations were made for the FPGA and associated connectors.

D. FCS20 System Operation

1. *FCS20 Boot Sequence and System Operation*

As detailed in Section II.A, the EC20 processor board consists of a TI DSP processor, an Altera Stratix FPGA, memory, clock driver circuits, and interface headers. The DSP gives the board its sequential processing power, and the FPGA performs all low-level and parallel interface functions to external components. A 32-bit, parallel data bus enables communication between the FPGA and the DSP. As the JTAG header (See Fig. 1) is only used for the initial DSP software load, the DSP communicates with external components solely through the FPGA.

The DSP serves as both the system's main sequential processor and as a configuration device for the FPGA. After power has been supplied to the board and the power supervisor has released the reset-line, the DSP will boot up and load the default application from Flash memory. Then, the DSP will configure the FPGA with an image from Flash via a dedicated 8-bit configuration data bus. After the FPGA image has been loaded and the DSP has released the FPGA hardware reset line, the DSP is able to communicate with the FPGA. At this point, the softcore CPU on the FPGA becomes the system master and initiates every flight control cycle by sending the most recent sensor data to the DSP, which then processes the data and ships the results back to the FPGA.

After the FPGA has become operational, it controls the flow of data within itself and to/from the DSP. It is responsible for loading updated DSP software into Flash. It also tells the DSP which application to run, gives software reset commands to the DSP, issues interrupts, and controls the DSP core frequency. It can also order its own reconfiguration, as illustrated in the following example:

1. As power is applied to the board, the DSP starts up and loads itself and the FPGA with the configuration currently stored in Flash memory. The softcore processor in the FPGA (the Nios$^{\text{TM}}$ CPU) then becomes the master controller.
2. A new version of the DSP application software is to be loaded. A request is sent to the master CPU in the FPGA and shortly thereafter, the new application is loaded into Flash memory via the DSP. Data may be sent to the FPGA through any of the 110 IO pins in any desirable format, but for this example we assume that the new application is loaded through a standard RS232 serial line to a UART in the Nios$^{\text{TM}}$ CPU.
3. The master CPU in the FPGA then orders the DSP to either perform a software reset or simply start execution of the new code.

2. *Asynchronous System Clocks*

The DSP, the FPGA, and the internal data bus on the EC20 all operate on separate clocks. The 20 MHz oscillator (see Fig. 1) drives two Phase Locked Loops (PLLs) in the DSP. The first is controlled dynamically up to 225 MHz and is used for the DSP core. The second is used for the internal bus, which runs at 50 MHz.

The FPGA receives the native 20 MHz (from the oscillator) and has multiple internal PLLs that generate frequencies for various components. In the standard configuration, two PLLs are used in the FPGA, though up to twelve can be supported. The FPGA also receives the bus clock signal. The asynchronous clock domains in the FPGA require special handling in the FIFOs. Nios$^{\text{TM}}$, the softcore CPU (in the FPGA), and its associated circuitry runs off one clock, and special dual-clock FIFOs are used to interface between Nios$^{\text{TM}}$ and the internal data bus. The advantage of this configuration is that clock frequencies for FPGA components and the internal data bus can be set independently from each other.

In summary, data from the SB20 sensor/power board is received in the EC20 through one of the 40 pin headers. It is then preprocessed in the FPGA and bundled and transferred to the DSP through a set of First-In First-Out (FIFO) queues in the FPGA. Actuator commands are then sent from the DSP back to the appropriate servo driver component in the FPGA, thus closing the main flight control loop.

3. *Power Requirements*

The FCS20 system power requirements are driven largely by the sum of the requirements for the DSP, the FPGA, and the sensors. When used in the minimum two-board FCS20 configuration, the SB20 is usually powered by a lithium polymer 11.1 Volt three-cell battery. Power in the system is split by four power regulators. 3.3 Volts is provided to both the FPGA and the DSP I/O on the EC20. 3.3 Volts is also used by the GPS and RS232 drivers on the SB20. The FPGA core requires 1.5 Volts and the DSP core requires 1.26 Volts. A linear regulator supplies 5 Volts to the inertial sensors and A/D converters.

Current consumption in the two-board configuration varies depending on the DSP clock rate and the number of FPGA Logic Cells used. At a clock speed of 225 MHz, the DSP core requires 700 mA, but at 100 MHz, the DSP requires only 100 mA. This provides a performance vs. operation-time option to the user. In general, however, the FCS20 operates at 1–3 Watts, 0.4 Watts of which is consumed by the SB20. An additional 1–3 Watts are required for each additional EC20 added to the system.

III. *Communications:* FCS20 Internal and External Data Flow

E. Internal Data Flow

Internal data flow in the FCS20 is dominated by the communication between the DSP and the FPGA that occurs through the high-speed 32-bit internal data bus. Multiple, parallel First-In First-Out (FIFO) components inside the FPGA enable communication with the DSP from dedicated components within the FPGA, avoiding information bottlenecks and reducing time delays. The FIFOs also act as buffers between the different components, which helps ensure data integrity.

Figure 4 highlights the main data flow pathways in the FCS20 system. Onboard the SB20, the analog sensors are read by the A/D converters. The accelerometers have both analog and digital outputs, and the digital outputs are sent directly to counters in the FPGA. The GPS on the SB20 communicates with the FPGA via an RS232 serial line, while the video encoder sends its output directly to an interface in the FPGA. As the FPGA receives packets of sensor data, it processes and packages the data, and sends relevant data to the DSP via the high-speed internal data bus. The data is then processed on the DSP by the guidance, navigation, and control algorithms. Control outputs are then sent back to the FPGA via the internal data bus, converted to PWM signals, and sent out to external servos. On the EC20, there is also communication between SDRAM #2 and the FPGA. SDRAM #2 is a memory block dedicated solely for use by the FPGA. SDRAM #1 is controlled by the DSP, and the Flash Memory is used to store the FPGA image and DSP software. Relevant data from the Nios^{TM} processor is also sent to an external communications device via a dedicated serial line. The FPGA allows for future flexibility through the addition of communication interface logic, image sensor interfaces, Ethernet, USB, and encryption or other protocols.

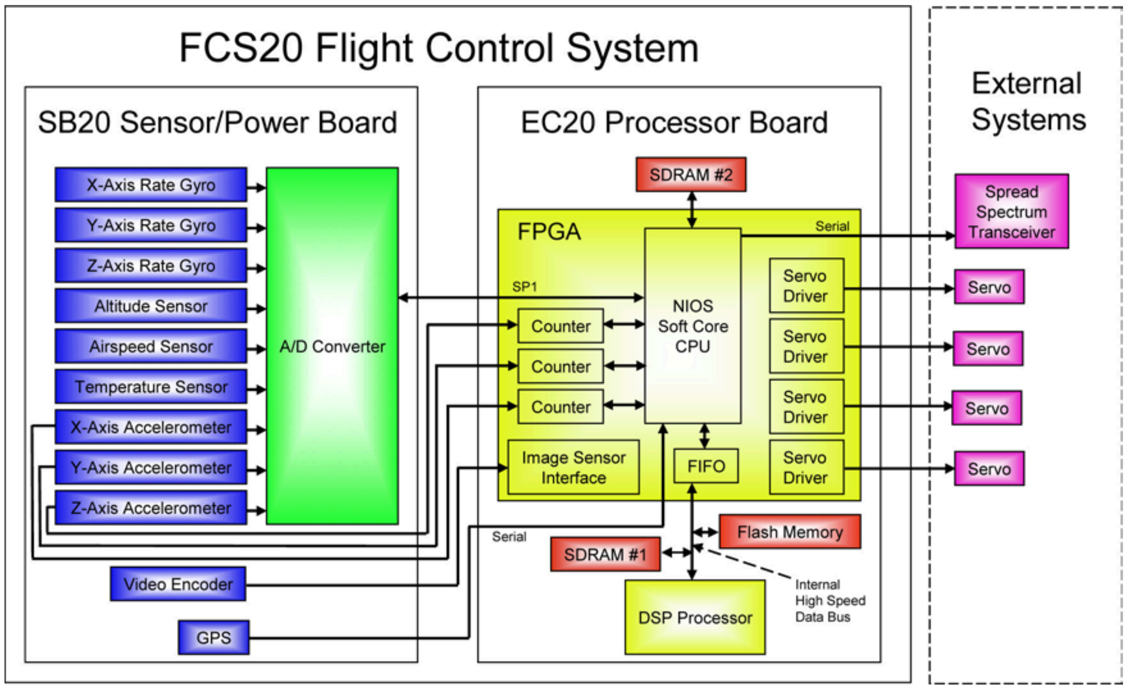

Fig. 4 FCS20 Data Flow Diagram.

F. Main and Secondary Bus Descriptions

The FCS20 has two main buses. The main bus is the 32-bit parallel high-speed internal data bus, with a maximum throughput of 250 Mbps, which acts as a conduit of information flow between the FPGA and the DSP. This very short internal bus runs directly between the DSP and the FPGA with just enough extra space to allow nearby placement of the bus accessible SDRAM and Flash memory (See Fig. 1 and Fig. 4).

The secondary bus is highly configurable and will talk only to the FPGA. The bus structure of this secondary bus was intentionally made flexible so that developers could choose a bus standard suitable for their application. The secondary bus will utilize high-speed differential serial channels from the FPGA and will be dedicated for board-to-board and other customized external communications. As configured, there are seven of these high-speed channels (rated for up to 840 Mbps) available on the 40 pin headers.

IV. *Information:* FCS20 Software Systems and Supported Algorithms

A. The FCS20 Operating System

The DSP in the FCS20 utilizes the MicroC/OS-II real-time operating system (RTOS). The operating system is a preemptive, multitasking kernel set up to manage up to 64 separate tasks. This kernel includes basic operating system services such as semaphores, mutual exclusion semaphores, event flags, message mailboxes, message queues, task management, fixed size memory block management, and time management functions.[8] The execution times for most services in the operating system are constant and deterministic and do not depend on the number of tasks running in the system. In the FCS20, the RTOS manages the data transfer to and from the FPGA.

B. The FCS20 Standard Vehicle Interface Library

The Standard Vehicle Interface Library (SVIL) is comprised of generic and highly capable data communication and simulation software modules that have been developed to support a large number of potential flight and simulator test configurations.[9] The lowest levels of the SVIL contain custom software modules that enable communication with different hardware components like sensors, actuators and physical communication ports. Every device driver in the SVIL is accompanied by a mathematical model of that sensor, including data packets as output by that sensor. This enables full simulation of the sensor during a variety of ground tests.

Seamless interfacing with platform independent algorithms is achieved by abstracting devices to the level of communication channels. To the algorithms, any device appears as a stream of packetized data on a particular communication channel. Communication packets in the system consist of a header and data. The header contains two synchronization bytes, a packet identifier, an indication of the packet size, and checksums for both the header and the data. Communication channels may be dynamically assigned at runtime and even rerouted during flight. Figure 5 depicts the design of the SVIL and illustrates that real and simulated devices can be mixed in any combination. For example, during laboratory work, where GPS signals are not available, simulated GPS data can be injected into the navigation system as though it is actual raw sensor data.

On many occasions, the failure or anomalous behavior of algorithms during a flight test is due to differences between the simulation setup and operational hardware. During flight, data passes through various components of hardware and software before it arrives at the navigation algorithm. Simulating these effects, which can include sampling, digitization, buffer overflows, too infrequent servicing of buffers etc. is important. A particular example occurs when sensor data from an inertial measurement unit (IMU) is available through a serial port, but the serial ring buffers are not sufficiently large to buffer more than one packet. If, at some instant, the device driver is unable to service the buffers fast enough, the ring buffer overwrites older but valid IMU packets. During flight, this phenomenon can introduce periodic or a-periodic losses of IMU packets. If a controller were developed in a simulation environment that did not account for this effect and then flight tested, vehicle behavior could result in a misdiagnosis of the problem. With the SVIL, many such problems are detected early because each device is essentially a channel, and during both simulation and development, data from the simulated device goes through the same device driver code and buffers at the same rate as it would during an actual flight.

Another important aspect of the SVIL is that in-flight recordings of sensor data channels can be played back to algorithms, effectively replicating the flight through the same flight code. Because the data is not just passively

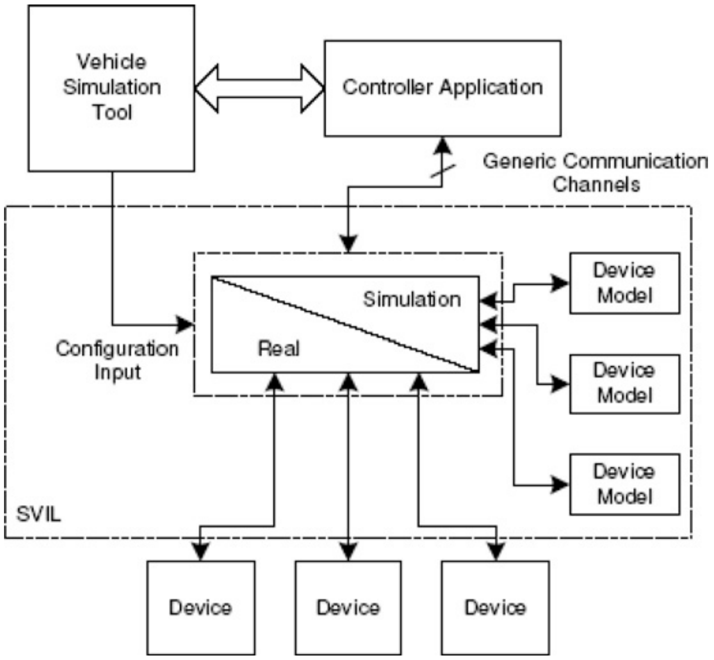

Fig. 5 Schematic of the Standard Vehicle Interface Library that depicts the abstraction from devices to Communication channels.

replayed, but rather is actively processed by the algorithms just as it would be during flight, this feature becomes an important diagnostic tool. Anomalous flight behaviors can be reproduced, the algorithms changed, and sensor data replayed until the problem is corrected, significantly reducing development and debugging time.

C. The Navigation Kalman Filter

An integral component of the FCS20 is the sixteen state Extended Kalman Filter (EKF) that uses data from the sensors on the SB20 processor board to generate a navigation solution that closely estimates the state of the system.[10−11] The EKF serves several important functions including: 1) estimating the orientation of the system from accelerations and angular rates, 2) removing process and measurement noise from the measurements, and 3) providing state estimates at 100 Hz, even though most sensor measurements are taken at a much slower rate.

1. The EKF Process Model

The EKF is described by a set of continuous and discrete equations. Equation (1) is the state vector for the EKF Process, and Equation (2) represents the continuous process model. The states in the navigation filter include: four quaternion components, three position states, three velocity states, and six accelerometer and gyro biases.

$$x(t) = [q_0, q_1, q_2, q_3, x_{pos}, y_{pos}, z_{pos}, u, v, w, b_{ax}, b_{ay}, b_{az}, b_{\omega x}, b_{\omega y}, b_{\omega z}] \tag{1}$$

$$\dot{x}(t) = f(t, x(t), u(t)) + w(t) \tag{2}$$

Since one of the main goals of the EKF is to provide values of the Euler orientation angles for pitch, roll, and yaw (or equivalently, in this case, the four quaternion parameters) based on acceleration and angular rate measurements, the process model f is made up of the following kinematic relationships [19]:

$$\begin{bmatrix} \dot{\hat{q}}_0 \\ \dot{\hat{q}}_1 \\ \dot{\hat{q}}_2 \\ \dot{\hat{q}}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -(\omega_{IMUx} - \hat{b}_p) & -(\omega_{IMUy} - \hat{b}_q) & -(\omega_{IMUz} - \hat{b}_r) \\ \omega_{IMUx} - \hat{b}_p & 0 & \omega_{IMUz} - \hat{b}_r & -(\omega_{IMUy} - \hat{b}_q) \\ \omega_{IMUy} - \hat{b}_p & -(\omega_{IMUz} - \hat{b}_r) & 0 & \omega_{IMUx} - \hat{b}_p \\ \omega_{IMUz} - b_r & \omega_{IMUy} - b_q & -(\omega_{IMUx} - b_p) & 0 \end{bmatrix} \begin{bmatrix} \hat{q}_0 \\ \hat{q}_1 \\ \hat{q}_2 \\ \hat{q}_3 \end{bmatrix}$$

$$\begin{aligned} \dot{\hat{x}} &= \hat{v} \\ \dot{\hat{v}} &= \hat{T}_{b->i}(a_{IMU} - \hat{b}_a) + g \\ \dot{\hat{b}}_a &= 0 \\ \dot{\hat{b}}_\omega &= 0 \end{aligned} \tag{3}$$

The transformation from the body frame to the inertial frame is given by:

$$T_{b->i} = \begin{bmatrix} 1 - 2(q_2^2 + q_3^2) & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & 1 - 2(q_1^2 + q_3^2) & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 - q_0 q_1) & 1 - 2(q_1^2 + q_2^2) \end{bmatrix} \tag{4}$$

Equations (5) and (6) represent the discrete measurement observations and the discrete output estimate. In these equations, time t_k is the time at which the k^{th} measurement is taken.

$$z_k = h_k(t_k, x_k, u_k) + v_k \tag{5}$$

$$\hat{y}_k = h_k(t_k, \hat{x}_k^+, u_k) \tag{6}$$

2. The EKF Measurement Model

The measurement model h corresponds to the sensor measurements (except for the IMU) at the c.g. The set of equations in (7) represent the sensor measurements corrected for the effects of measurement noise and for their off-center location. In Equation (8), z_k is the vector of corrected measurements that are available to the EKF.

$$\begin{aligned} x_{poscg} &= x_{posGPS} - T_{b->i} r_{GPS} - w_x \\ v_{cg} &= v_{GPS} - T_{b->i} \omega \times r_{GPS} - w_v \\ h_{cg} &= -h_{sensor} - T_{b->i}[3] r_{sensor} + h_t - w_{sensor} \end{aligned} \tag{7}$$

$$z_k = [x_{poscg} v_{cg} h_{cg}] \tag{8}$$

3. The Extended Kalman Filter Implementation

Equations (9) and (10) are the state estimate time update (predictor) equations, and Equations (11) and (12) are the error covariance time update equations. In these equations, the continuous derivatives (Equations (9) and (11)) are used along with standard numerical integration methods to propagate forward in time. In the case of the state estimate propagation, a 2nd order modified Euler's method (Equation (10)) is used, where $\dot{\hat{x}}_{k+1}^-$ is determined using a temporary value for $\hat{x}_{k+1}^-$ that comes from a 1st order Euler predictor step. In the case of the error covariance, a 1st order Euler's method (Equation (12)) is used.

$$\dot{\hat{x}}(t) = f(t, \hat{x}_k^+, u) \tag{9}$$

$$\hat{x}_{k+1}^- = \hat{x}_k^+ + \frac{\Delta t}{2}(\dot{\hat{x}}_{k+1}^- + \dot{\hat{x}}_k) \tag{10}$$

$$\dot{P}(t_k) = F(t, \hat{x}_k^-, u_k) P_k^- + P_k^- F^T(t, \hat{x}_k^-, u_k) + Q(t_k) \tag{11}$$

$$P_{k+1}^- = P_k^+ + \dot{P}(t_k) \Delta t \tag{12}$$

Equation (13) is the equation for the Kalman Filter gain, and Equations (14) and (15) are the state and error covariance measurement update (corrector) equations.

$$K_k = P_k^- H_k^T [H_k P_k^- H_k^T + R_k]^{-1} \tag{13}$$

$$\hat{x}_k^+ = \hat{x}_k^- + K_k(z_k - h_k(t_k, \hat{x}_k^-, u_k)) \tag{14}$$

$$P_k^+ = (I - K_k H_k) P_k^- \tag{15}$$

F_k, and H_k, from Equations (11), (13), and (15), are calculated as follows:

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{x=\hat{x}_k^-} \tag{16}$$

$$H_k = \left.\frac{\partial h_k}{\partial x}\right|_{x=\hat{x}_k^-} \tag{17}$$

$$F_k = \begin{bmatrix} F_{11} & 0 & 0 & 0 & F_{15} \\ 0 & 0 & I_{3\times3} & 0 & 0 \\ F_{31} & 0 & 0 & F_{34} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad H_k = \begin{bmatrix} I_{3\times3} & 0 \\ 0 & I_{3\times3} \end{bmatrix} \tag{18}$$

Where:

$$F_{11} = \frac{1}{2}\begin{bmatrix} 0 & \hat{b}_p - \omega_{IMUx} & \hat{b}_q - \omega_{IMUy} & \hat{b}_r - \omega_{IMUz} \\ -\hat{b}_p + \omega_{IMUx} & 0 & -\hat{b}_r + \omega_{IMUz} & \hat{b}_{\mathrm{q}} - \omega_{IMUy} \\ -\hat{b}_q + \omega_{IMUy} & \hat{b}_r - \omega_{IMUz} & 0 & -\hat{b}_p + \omega_{IMUx} \\ -\hat{b}_r + \omega_{IMUz} & -\hat{b}_{\mathrm{q}} + \omega_{IMUy} & \hat{b}_p - \omega_{IMUx} & 0 \end{bmatrix} \tag{19}$$

$$F_{15} = \frac{1}{2}\begin{bmatrix} \hat{q}_1 & \hat{q}_2 & \hat{q}_3 \\ -\hat{q}_0 & \hat{q}_3 & -\hat{q}_2 \\ -\hat{q}_3 & -\hat{q}_0 & \hat{q}_1 \\ \hat{q}_2 & -\hat{q}_1 & -\hat{q}_0 \end{bmatrix} \tag{20}$$

$$I_{3\times3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{21}$$

$$F_{31} = \begin{bmatrix} 2(\hat{q}_3\alpha_y - \hat{q}_2\alpha_z) & -2(\hat{q}_2\alpha_y + \hat{q}_3\alpha_z) & 2(2\hat{q}_2\alpha_x - \hat{q}_1\alpha_y - \hat{q}_0\alpha_z) & 2(2\hat{q}_3\alpha_x + \hat{q}_0\alpha_y - \hat{q}_1\alpha_z) \\ 2(\hat{q}_1\alpha_z - \hat{q}_3\alpha_x) & 2(2\hat{q}_1\alpha_y - \hat{q}_2\alpha_x + \hat{q}_0\alpha_z) & -2(\hat{q}_1\alpha_x + \hat{q}_3\alpha_z) & 2(2\hat{q}_3\alpha_y - \hat{q}_0\alpha_x - \hat{q}_2\alpha_z) \\ 2(\hat{q}_2\alpha_x - \hat{q}_1\alpha_y) & 2(2\hat{q}_1\alpha_z - \hat{q}_3\alpha_x - \hat{q}_0\alpha_y) & 2(2\hat{q}_2\alpha_z - \hat{q}_3\alpha_y + \hat{q}_0\alpha_x) & -2(\hat{q}_1\alpha_x + \hat{q}_2\alpha_y) \end{bmatrix} \tag{22}$$

Where:

$$\begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{bmatrix} = \begin{bmatrix} \hat{b}_{ax} - a_{IMUx} \\ \hat{b}_{ay} - a_{IMUy} \\ \hat{b}_{az} - a_{IMUz} \end{bmatrix} \tag{23}$$

$$F_{34} = \begin{bmatrix} -1 + 2(\hat{q}_2^2 + \hat{q}_3^2) & -2(\hat{q}_1\hat{q}_2 - \hat{q}_0\hat{q}_3) & -2(\hat{q}_1\hat{q}_3 + \hat{q}_0\hat{q}_2) \\ -2(\hat{q}_0\hat{q}_3 + \hat{q}_1\hat{q}_2) & -1 + 2(\hat{q}_1^2 + \hat{q}_3^2) & -2(\hat{q}_2\hat{q}_3 - \hat{q}_0\hat{q}_1) \\ -2(\hat{q}_1\hat{q}_3 - \hat{q}_0\hat{q}_2) & -2(\hat{q}_0\hat{q}_1 + \hat{q}_2\hat{q}_3) & -1 + 2(\hat{q}_1^2 + \hat{q}_2^2) \end{bmatrix} \tag{24}$$

Equations (19) and (20) are generated by taking the partial derivatives of the quaternion component equations in Equation (1) with respect to the quaternion components and gyro biases, respectively. Equations (22) and (24) are generated by taking the partial derivatives of the velocity equations in (3) with respect to the quaternion components and accelerometer biases, respectively. The EKF equations (9)–(15) above are similar to those derived in.[12–13]

It is interesting to note that while one of the main functions of the EKF is to estimate orientation angles based on rates and accelerations, the rates and accelerations are not included as process states, nor are they included in the EKF measurements in z_k (Equation (8)). The acceleration and rate measurements are corrected for being located off the c.g. in Equations (25)–(26), and then are used directly in the right hand side of the process equations (Equations (3)). This is how the process noise is introduced into the EKF system. Another similar EKF implementation can be found in[14] for comparison.

The IMU (which is comprised of the accelerometers and rate gyros) gives the 3-axis acceleration, a_{IMU}, and angular rate, ω_{IMU}, of the point at which it is mounted (which is typically not exactly at the c.g.). These measurements are expressed in the body frame and are true values perturbed by two effects, a bias and a measurement noise, as follows:

$$a_{cg} = a_{IMU} - \dot{\omega} \times r_{IMU} - \omega \times (\omega \times r_{IMU}) + b_a + w_a \tag{25}$$

$$\omega_{cg} = \omega_{IMU} + b_\omega + w_\omega \tag{26}$$

The GPS gives the position x_{posGPS} and the velocity v_{GPS} of the mounting location of the sensor. These measurements are expressed in the NED frame and are true values perturbed by two effects: measurement noise and latency (i.e. the output provided at time t corresponds to a measurement made at time t-latency). The EKF uses these measurements, corrected for the effects of offset from the center of gravity and measurement noise, as follows:

$$x_{poscg} = x_{posGPS} - \hat{T}_{b->i} r_{GPS} = x_{posGPS} + (T_{b->i} - \hat{T}_{b->i}) r_{GPS} + w_x \tag{27}$$

$$v_{cg} = v_{GPS} - T_{b->i}\omega \times r_{GPS} = v_{GPS} + (T_{b->i}\omega \times r_{GPS} - \hat{T}_{b->i}\hat{\omega} \times r_{GPS}) + w_v \tag{28}$$

$$\hat{\omega} = \omega_{IMU} - \hat{b}\omega = \omega_{IMU} + b_\omega - \hat{b}_\omega + w_\omega \tag{29}$$

The latency in the GPS measurements is handled by making the assumption that the latency is small enough that the correction for the measurement will not be significantly different for the current state versus the state at which the measurement actually happened. In the actual implementation, a buffer is maintained that stores a history of estimated states, so that when the residuals are calculated, the measurement is compared with the estimated state at the time the measurement occurred.

4. *The Measurement Noise Covariance Matrix*

The measurement noise covariance matrix R_k has the form:

$$R_k = \begin{bmatrix} R_{11} & 0 \\ 0 & R_{22} \end{bmatrix} \tag{30}$$

where R_{11} corresponds to the measurement noise of the GPS position, R_{22} corresponds to the measurement noise of the GPS velocity. The assumption is made that $T_{b->i} \cong \hat{T}_{b->i}$ in Equation (27), and thus the measurement noise covariance matrix on x_{poscg} is simply:

$$R_{11} = E(w_x\, w_x^T) = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_z^2 \end{bmatrix} \tag{31}$$

The measurement noise covariance on the velocity, v, is more complex. We first let $\tilde{b}_\omega = b_\omega - \hat{b}_\omega$ be the error in estimating the gyro biases. We then assume that $\tilde{b}_\omega$, w_ω, and w_v are independent and also that $E(\tilde{b}_\omega\, \tilde{b}_\omega^T)$ is quasi-diagonal (i.e. cross correlations are small relative to the autocorrelations). Then the measurement noise covariance matrix on v is given by:

$$
\begin{aligned}
R_{22} &= E(w_v w_v^T) + \hat{T}_{b->i} E((\tilde{b}_\omega \times r_{GPS})(\tilde{b}_\omega \times r_{GPS})^T)\hat{T}_{b->i}^T + \hat{T}_{b->i} E((w_\omega \times r_{GPS})(w_\omega \times r_{GPS})^T)\hat{T}_{b->i}^T \\
&= \sigma_v^2 I_3 + \hat{T}_{b->i} \left(\begin{bmatrix} P_{15}z^2 + P_{16}y^2 & -P_{16}xy & -P_{15}xy \\ -P_{16}xy & P_{16}x^2 + P_{14}z^2 & -P_{14}yz \\ -P_{15}xy & -P_{14}yz & P_{15}x^2 + P_{14}y^2 \end{bmatrix} \right. \\
&\quad \left. + \sigma_\omega^2 \begin{bmatrix} z^2 + y^2 & -xy & -xz \\ -xy & x^2 + z^2 & -yz \\ -xz & -yz & x^2 + y^2 \end{bmatrix} \right) \hat{T}_{b->i}^T
\end{aligned} \tag{32}
$$

where x, y, and z are the coordinates of the GPS in the body frame, and P_j is the j^{th} diagonal element of the error covariance matrix. The off diagonal terms in R_{22} are due mainly to the offset of the GPS antenna from the c.g., and these terms are neglected, resulting in a diagonal measurement noise covariance matrix.

5. The Process Noise Covariance Matrix

As mentioned earlier, the process noise is introduced to the EKF process by the IMU measurements. Before being used, the IMU measurements are corrected for mounting location by:

$$
\begin{aligned}
a_{corr} &= a_{IMU} - \dot{\hat{\omega}} \times r_{IMU} - \hat{\omega} \times (\hat{\omega} \times r_{IMU}) \\
\hat{\omega} &= \omega_{IMU} - \hat{b}_\omega \\
\dot{\hat{\omega}} &= \frac{\hat{\omega}(t + \Delta t) - \hat{\omega}(t)}{\Delta t}
\end{aligned} \tag{33}
$$

So now:

$$
\begin{aligned}
\dot{\hat{v}} &= \hat{T}_{b->i}(a_{corr} - \hat{b}_a) + g = \hat{T}_{b->i}(a_{cg} + (\dot{\omega} - \dot{\hat{\omega}}) \times r_{IMU} + \omega \times (\omega \times r_{IMU}) \\
&\quad - \hat{\omega} \times (\hat{\omega} \times r_{IMU}) + b_a - \hat{b}_a + w_a) + g
\end{aligned} \tag{34}
$$

The process noise for acceleration is assumed constant in the body axes, so the noise covariance is simply $\sigma_{acc}^2 I_{3\times 3}$. Now, considering the equation for the quaternion derivative

$$
\begin{aligned}
\begin{bmatrix} \dot{\hat{q}}_0 \\ \dot{\hat{q}}_1 \\ \dot{\hat{q}}_2 \\ \dot{\hat{q}}_3 \end{bmatrix} &= \frac{1}{2} \begin{bmatrix} 0 & -(\omega_{IMUx} - \hat{b}_p) & -(\omega_{IMUy} - \hat{b}_q) & -(\omega_{IMUz} - \hat{b}_r) \\ \omega_{IMUx} - \hat{b}_p & 0 & \omega_{IMUz} - \hat{b}_r & -(\omega_{\text{IMUy}} - \hat{b}_q) \\ \omega_{IMUy} - \hat{b}_p & -(\omega_{IMUz} - \hat{b}_r) & 0 & \omega_{IMUx} - \hat{b}_p \\ \omega_{IMUz} - b_r & \omega_{IMUy} - b_q & -(\omega_{IMUx} - b_p) & 0 \end{bmatrix} \begin{bmatrix} \hat{q}_0 \\ \hat{q}_1 \\ \hat{q}_2 \\ \hat{q}_3 \end{bmatrix} \\
&= \frac{1}{2} \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \hat{q} + \frac{1}{2} \begin{bmatrix} 0 & -\tilde{b}_p & -\tilde{b}_q & -\tilde{b}_r \\ \tilde{b}_p & 0 & \tilde{b}_r & -\tilde{b}_q \\ \tilde{b}_q & -\tilde{b}_r & 0 & \tilde{b}_p \\ \tilde{b}_r & \tilde{b}_q & -\tilde{b}_p & 0 \end{bmatrix} \hat{q} + \frac{1}{2} \begin{bmatrix} 0 & -w_p & -w_q & -w_r \\ w_p & 0 & w_r & -w_q \\ w_q & -w_r & 0 & w_p \\ w_r & w_q & -w_p & 0 \end{bmatrix} \hat{q}
\end{aligned} \tag{35}
$$

and again, assuming that $E(\tilde{b}_\omega \, \tilde{b}_\omega^t)$ is quasi diagonal, and that $\tilde{b}_\omega$ and w_ω are independent, the process noise covariance $Q(t)$ has the following form:

$$\frac{\sigma_\omega^2}{4}\begin{bmatrix} 1-\hat{q}_0^2 & -\hat{q}_0\hat{q}_1 & -\hat{q}_0\hat{q}_2 & -\hat{q}_0\hat{q}_3 \\ -\hat{q}_0\hat{q}_1 & 1-\hat{q}_1^2 & -\hat{q}_1\hat{q}_2 & -\hat{q}_1\hat{q}_3 \\ -\hat{q}_0\hat{q}_2 & -\hat{q}_1\hat{q}_2 & 1-\hat{q}_2^2 & -\hat{q}_2\hat{q}_3 \\ -\hat{q}_0\hat{q}_3 & -\hat{q}_1\hat{q}_3 & -\hat{q}_2\hat{q}_3 & 1-\hat{q}_3^2 \end{bmatrix}$$

$$+\frac{1}{2}\begin{bmatrix} P_{14}\hat{q}_1^2+P_{15}\hat{q}_2^2+P_{16}\hat{q}_3^2 & -P_{14}\hat{q}_0\hat{q}_1+P_{15}\hat{q}_2\hat{q}_3-P_{16}\hat{q}_2\hat{q}_3 & -P_{14}\hat{q}_1\hat{q}_3-P_{15}\hat{q}_0\hat{q}_2+P_{16}\hat{q}_1\hat{q}_3 & P_{14}\hat{q}_1\hat{q}_2-P_{15}\hat{q}_1\hat{q}_2-P_{16}\hat{q}_0\hat{q}_3 \\ -P_{14}\hat{q}_0\hat{q}_1+P_{15}\hat{q}_2\hat{q}_3-P_{16}\hat{q}_2\hat{q}_3 & P_{14}\hat{q}_0^2+P_{15}\hat{q}_3^2+P_{16}\hat{q}_2^2 & -P_{14}\hat{q}_0\hat{q}_3+P_{15}\hat{q}_0\hat{q}_3-P_{16}\hat{q}_1\hat{q}_2 & -P_{14}\hat{q}_0\hat{q}_2+P_{15}\hat{q}_1\hat{q}_3-P_{16}\hat{q}_0\hat{q}_2 \\ -P_{14}\hat{q}_1\hat{q}_3-P_{15}\hat{q}_0\hat{q}_2+P_{16}\hat{q}_1\hat{q}_3 & -P_{14}\hat{q}_0\hat{q}_3+P_{15}\hat{q}_0\hat{q}_3-P_{16}\hat{q}_1\hat{q}_2 & P_{14}\hat{q}_3^2+P_{15}\hat{q}_0^2+P_{16}\hat{q}_1^2 & -P_{14}\hat{q}_2\hat{q}_3+P_{15}\hat{q}_0\hat{q}_1-P_{16}\hat{q}_0\hat{q}_1 \\ P_{14}\hat{q}_1\hat{q}_2-P_{15}\hat{q}_1\hat{q}_2-P_{16}\hat{q}_0\hat{q}_3 & -P_{14}\hat{q}_0\hat{q}_2+P_{15}\hat{q}_1\hat{q}_3-P_{16}\hat{q}_0\hat{q}_2 & -P_{14}\hat{q}_2\hat{q}_3+P_{15}\hat{q}_0\hat{q}_1-P_{16}\hat{q}_0\hat{q}_1 & P_{14}\hat{q}_2^2+P_{15}\hat{q}_1^2+P_{16}\hat{q}_0^2 \end{bmatrix} \tag{36}$$

This is effectively the result of starting with a process noise covariance of $\sigma_\omega^2 I_{4\times4}$ and zeroing the components along the norm of the quaternion (since we have perfect information about the norm). In practice, however, this form for the process noise covariance is singular, so we use $\sigma_\omega^2 I_{4\times4}$ directly.

D. The Guidance and Neural Net Based Adaptive Control Systems

Figure 6 shows the architecture of the neural network adaptive controller and the Baseline Trajectory Generator that acts as the guidance system. The Neural Adaptive Controller (NAC) is an adaptive neural network trajectory-following controller with 18 neural network inputs, 5 hidden layer neurons, and 6 outputs for each of the 6 degrees of freedom.[15–16] The NAC has been extensively flight tested with a wide range of applications including autonomous takeoffs, landings, and trajectory tracking.[17] Furthermore, the stability of the system has been rigorously addressed.[16] A key advantage of the NAC is its ability to accommodate changing dynamics and payload configurations without having to rely on substantial system identification efforts.[16] This feature has been demonstrated by applying the architecture to flight tests of the GTMax research UAV,[17] and the ducted fan powered GTSpy UAV.[18]

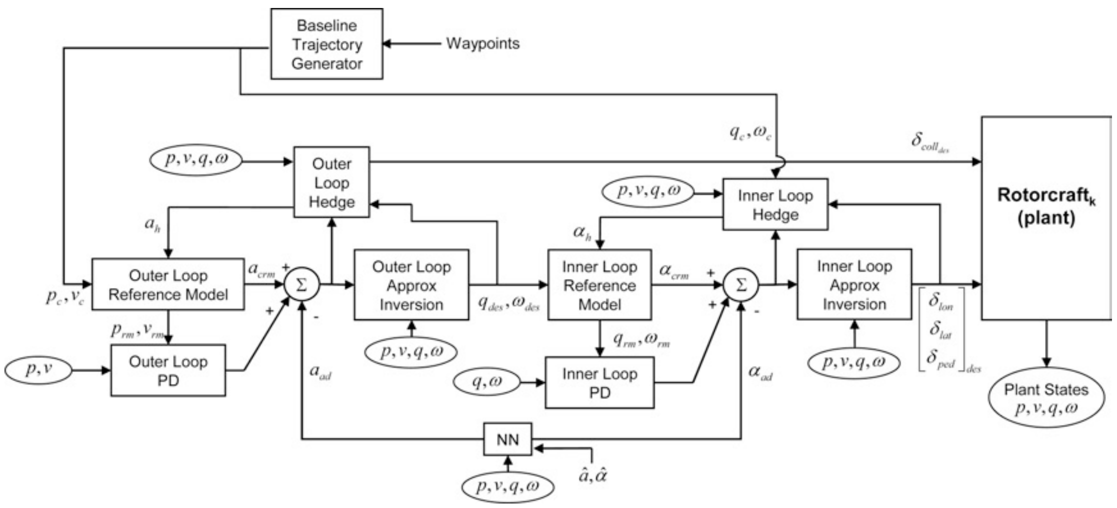

Fig. 6 The Neural Adaptive Guidance and Control System Architecture.

In Fig. 6, the Outer Loop Approximate Inversion is created by first generating a linear model of the aircraft around a trim condition while neglecting the coupling between the attitude and translational dynamics.[16] Choosing an invertible control matrix allows for the determination of a linear inverse control law. The outputs of the outer loop inverse law represent the desired values of attitude and angular rate, which are fed to the Inner Loop Reference Model and desired control command for the collective.[16] The Inner Loop Approximate Inversion is based around an invertible, linear, point-mass helicopter model. The gains of the PD linear compensators are chosen to place the poles of the closed loop system's longitudinal dynamics (inner loop PD affects the pitch angle, and outer loop PD affects the fore-aft position) such that the result is favorable longitudinal dynamic closed-loop behavior[16]. The reference models are designed with desirable dynamic characteristics, and they neglect actuator dynamics. The inner and outer loop hedge blocks are used to prevent the adaptive element from trying to adapt to selected system input characteristics.

V. Results Generated with the FCS20 Guidance, Navigation, and Control System

E. Simulation-Based Results

The following series of plots were generated using: 1) the Standard Vehicle Interface Library (SVIL) described in Section IV.B, 2) the navigation Extended Kalman Filter (EKF) described in Section IV.C, 3) the Guidance and Control Algorithms described in Section IV.D, and 4) a simulation of Georgia Tech's Helispy (GTSpy) 11-inch ducted-fan UAV (Fig. 7). Figures 9, 10, and 11 demonstrate that the EKF correctly estimates the orientation of the system from accelerations and angular rates. In these three plots, the estimated pose angles closely follow the actual angles and do not drift or diverge. Figures 12, 13, and 14 show that the EKF is removing process and measurement noise from the measurements. Notice that in these three figures, the estimated values are smoother and much closer to the actual values than the measured values are. Figure 14 shows that the EKF is providing state estimates at 100 Hz even though most sensor measurements are taken at a much slower rate. Finally, Fig. 15 demonstrates that the guidance system and Neural Adaptive Controller (NAC) are functioning properly to allow the GTSpy to traverse the desired "box" pattern.

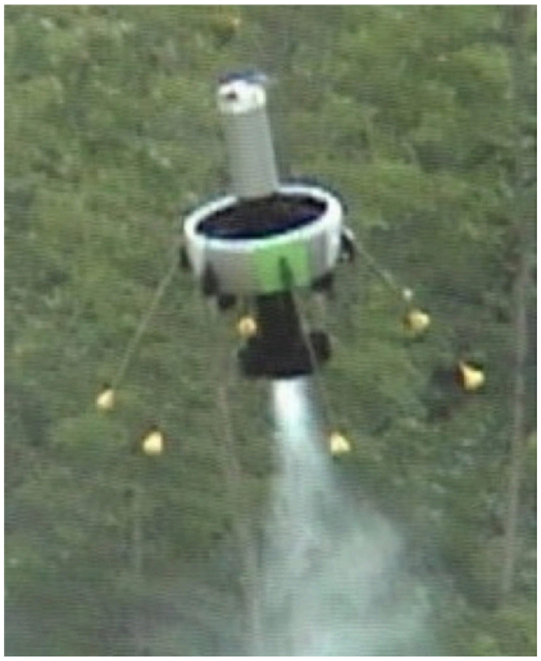

Fig. 7 Georgia Tech's HeliSpy (GTSpy) 11 inch ducted-fan UAV.

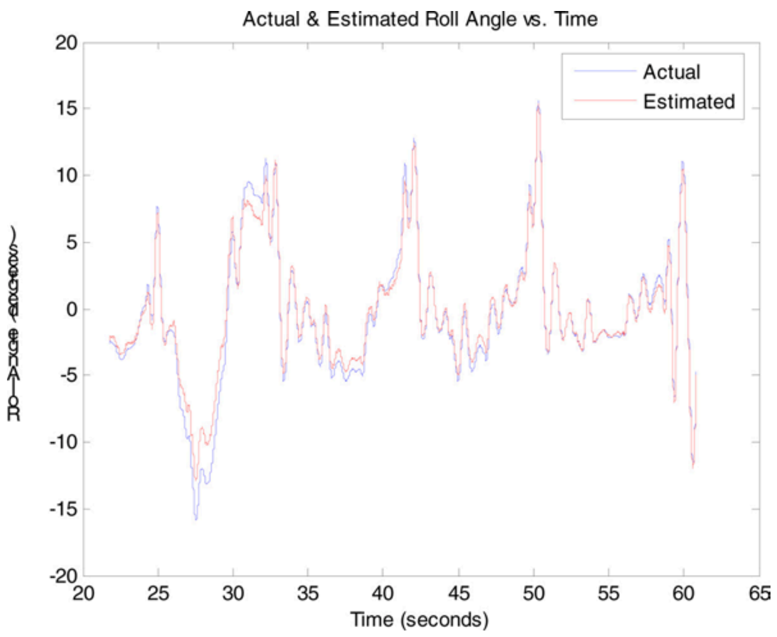

Fig. 8 Actual and Estimated Roll Angle vs. Time.

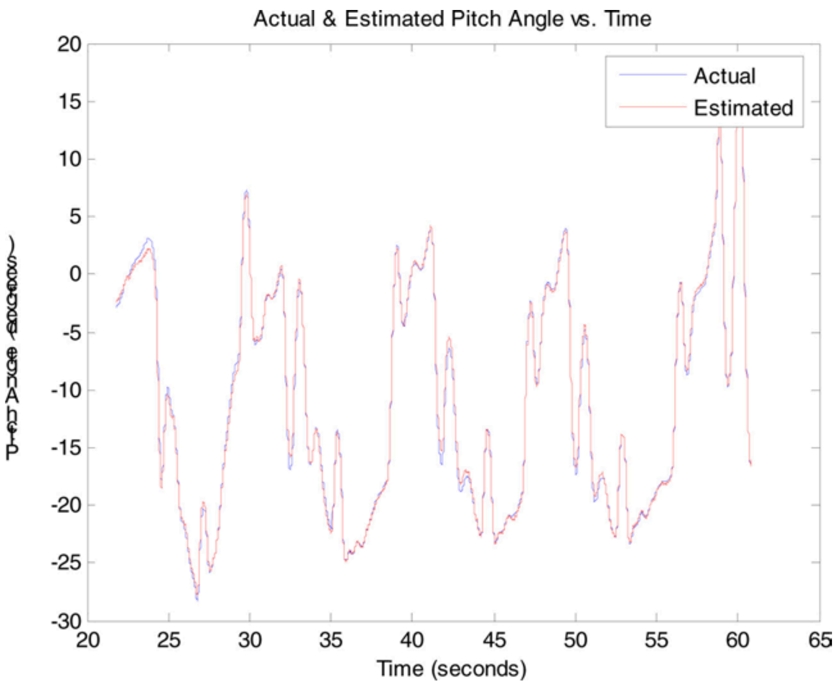

Fig. 9 Actual and Estimated Pitch Angle vs. Time.

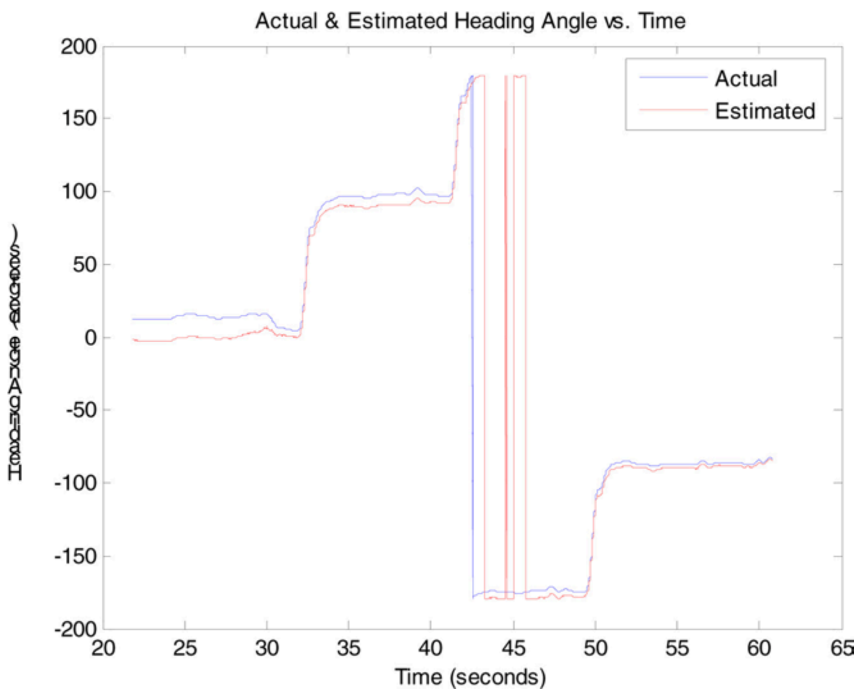

Fig. 10 Actual and Estimated Heading Angle vs. Time.

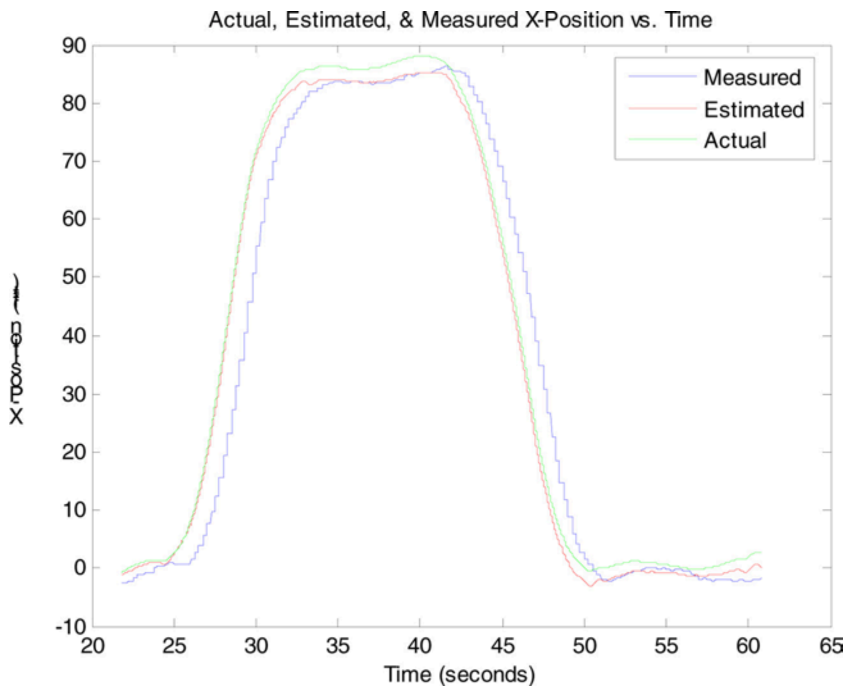

Fig. 11 Actual, Estimated, and Measured X-Position vs. Time.

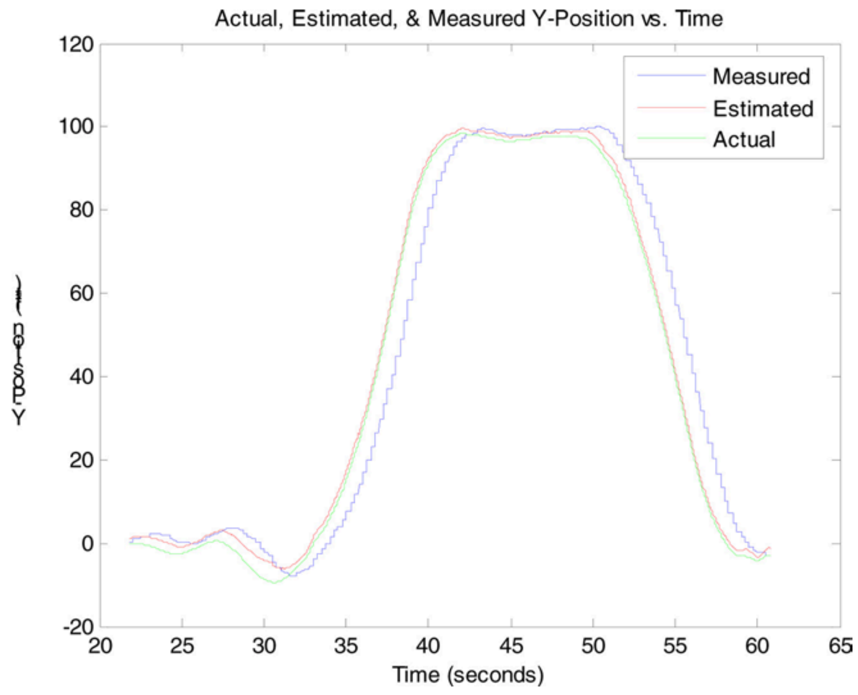

Fig. 12 Actual, Estimated, and Measured Y-Position vs. Time.

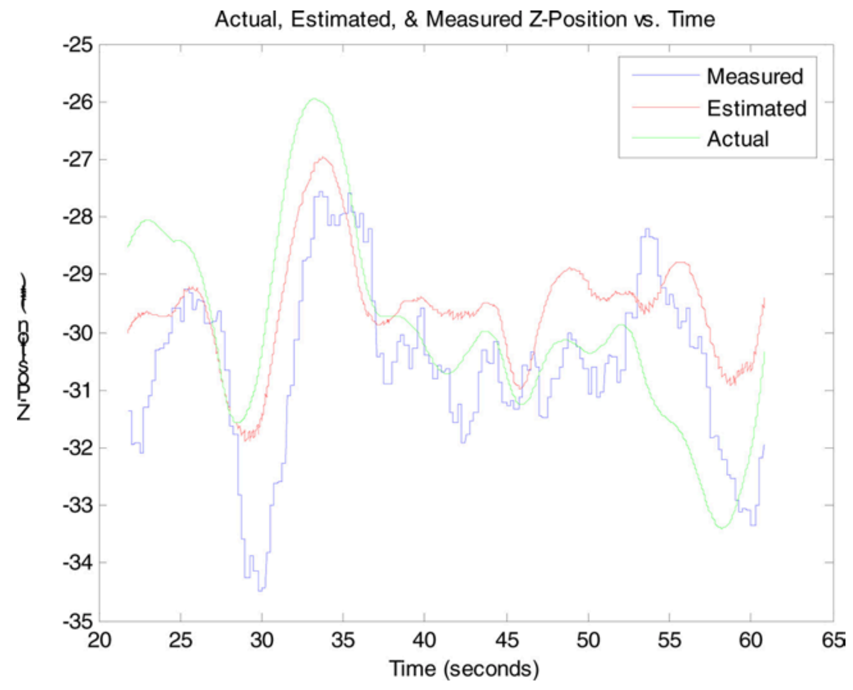

Fig. 13 Actual, Estimated, and Measured Z-Position vs. Time.

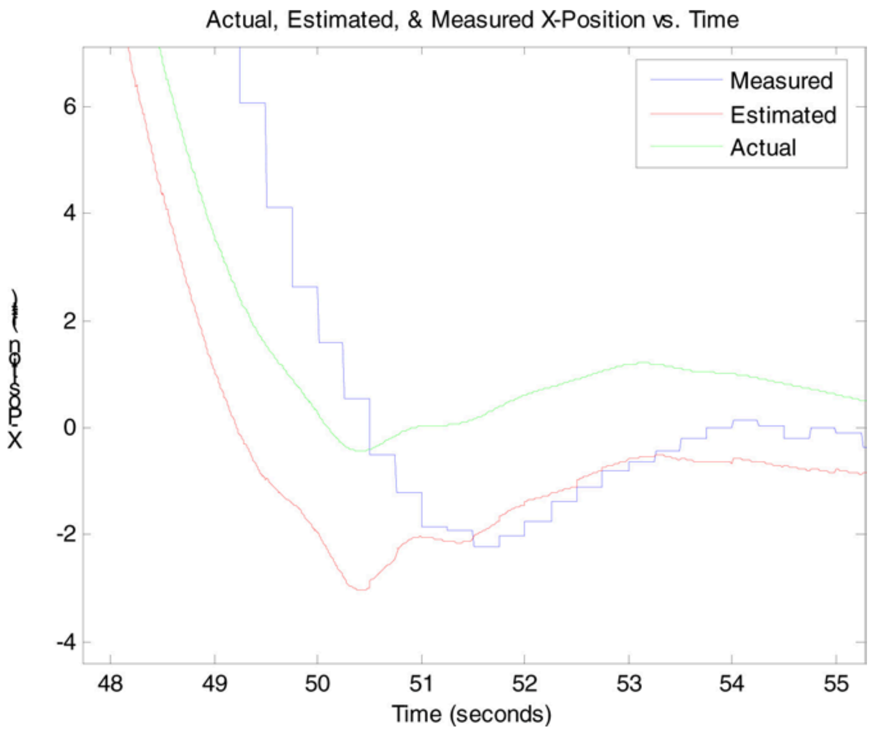

Fig. 14 Actual, Estimated, and Measured X-Position vs. Time.

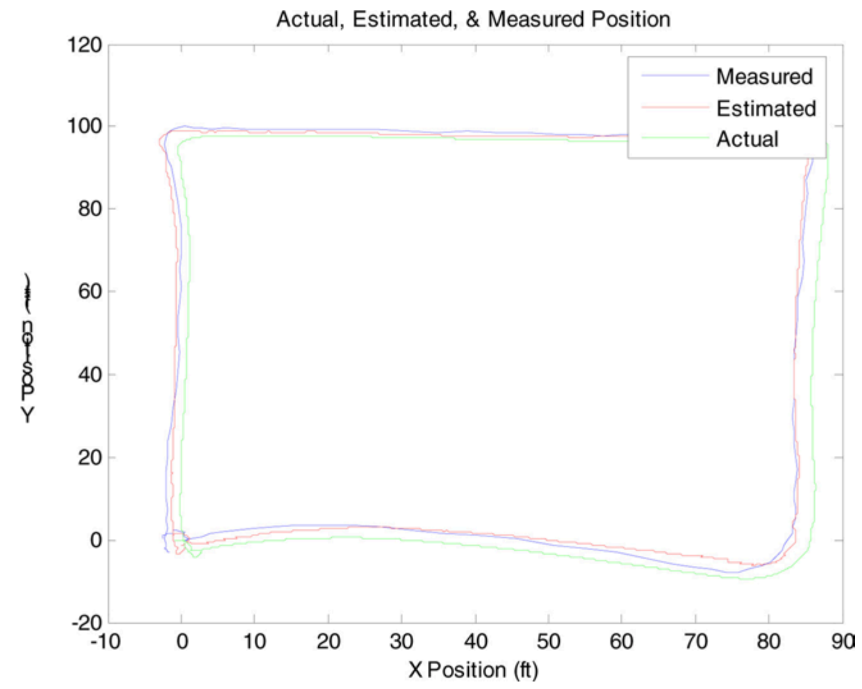

Fig. 15 Actual, Estimated, and Measured Position.

F. Flight-Test Results

Following initial checkout on Georgia Tech's GTMax UAV,[17] the FCS20 was utilized for guidance, navigation, and control during flight tests of a GTSpy 11 inch ducted-fan UAV[18] (Fig. 7). A number of tethered and approximately 10 un-tethered flights using the FCS20 have been conducted.

Results from a flight test in which the GTSpy autonomously executes a box maneuver similar to the one described in Section V.A are shown in Figs 17 through 20. A video clip of this flight can be seen here.

In the flight test maneuver, the box was significantly smaller than in the simulated case. Also, in the flight test maneuver, the aircraft had a zero degree commanded heading throughout the maneuver, whereas in the simulated case, the commanded heading was in the direction of the next waypoint. As expected from the simulation results, the controller performs well during the maneuver. However, comparison of Fig. 15 with Fig. 16 shows that the controller does not track the trajectory as closely as in the simulated case. This is the result of two effects that were not included in the above simulation runs: quantization error on the sensor measurements and atmospheric disturbances. The flight test was performed with light, intermittent wind, whose presence contributed to the decreased performance. There is one large discrepancy occurring on the third leg of the maneuver (See Fig. 16). This is probably due to either a larger gust of wind or a large, erroneous GPS position measurement. Figure 17 shows the heading tracking performance. The heading varies from the desired value more than in the simulation but remains within reasonable bounds. Figures 18 and 19 show the position tracking with respect to the body frame axes. Some of the observed errors are due to differences between the commanded and actual attitude. Additionally, some of the large errors noted at times close to 20 seconds are due to the discrepancy observed in the third leg. An additional video clip of the GTSpy being air launched from the GTMax can be seen here. and a video clip of the GTSpy performing a smooth landing can be seen here.

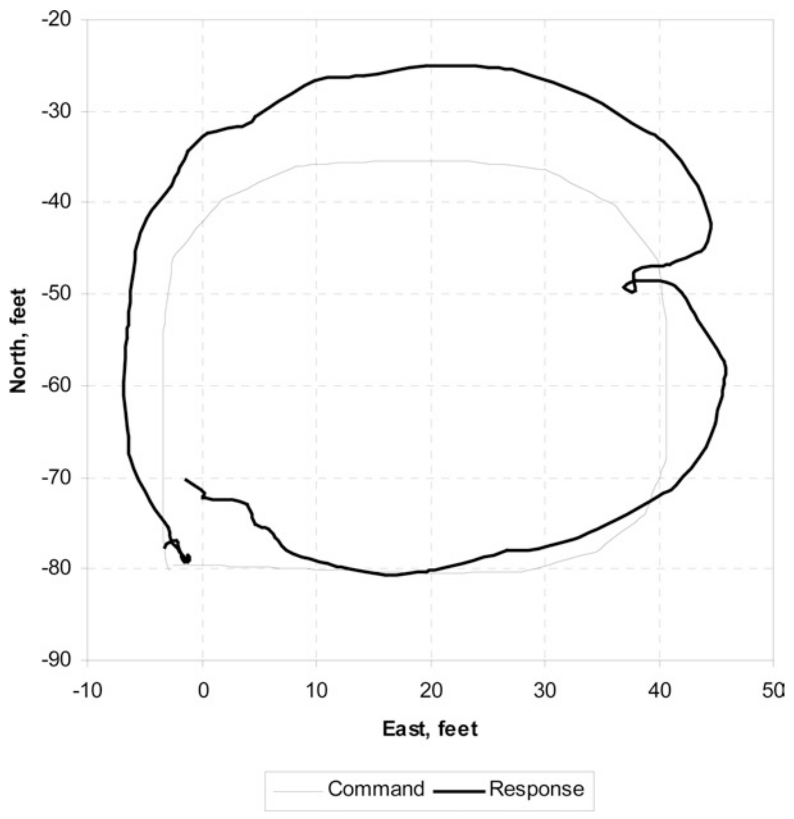

Fig. 16 Flight Test Position Results from Box Maneuver.

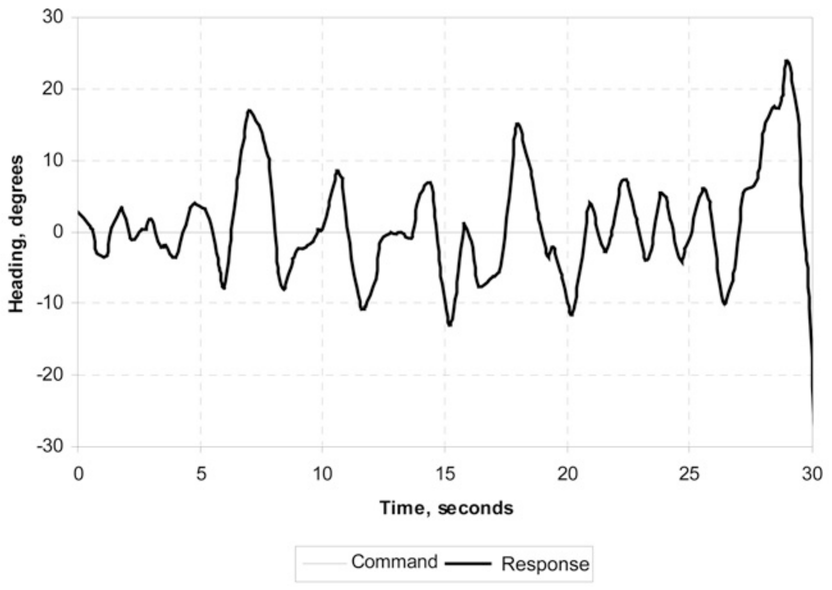

Fig. 17 Flight Test Heading Results from Box Maneuver.

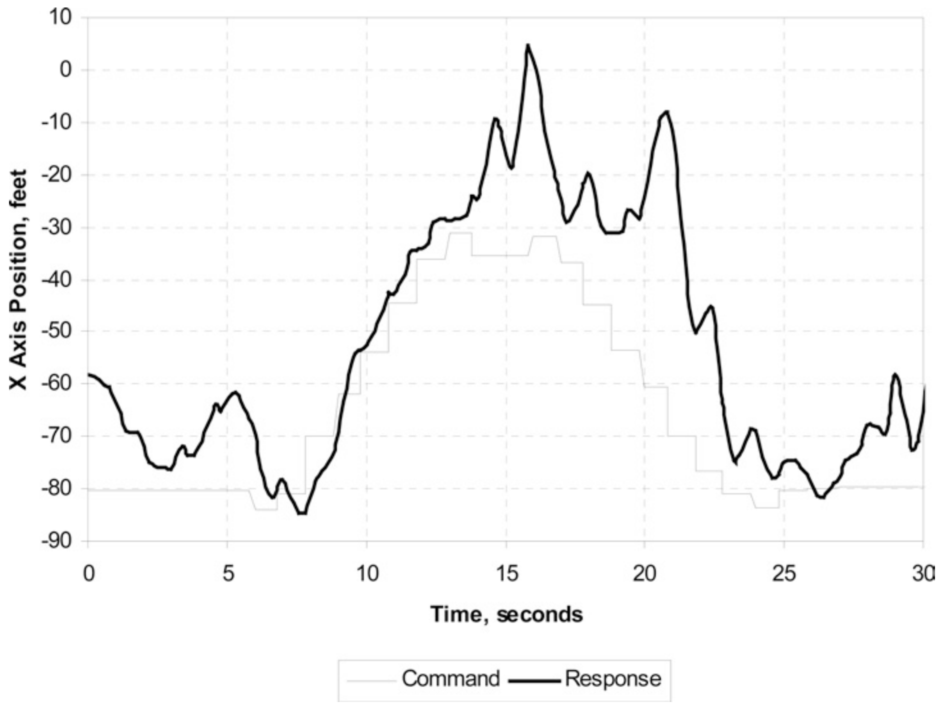

Fig. 18 Flight Test X-Body Axis Position from Box Maneuver.

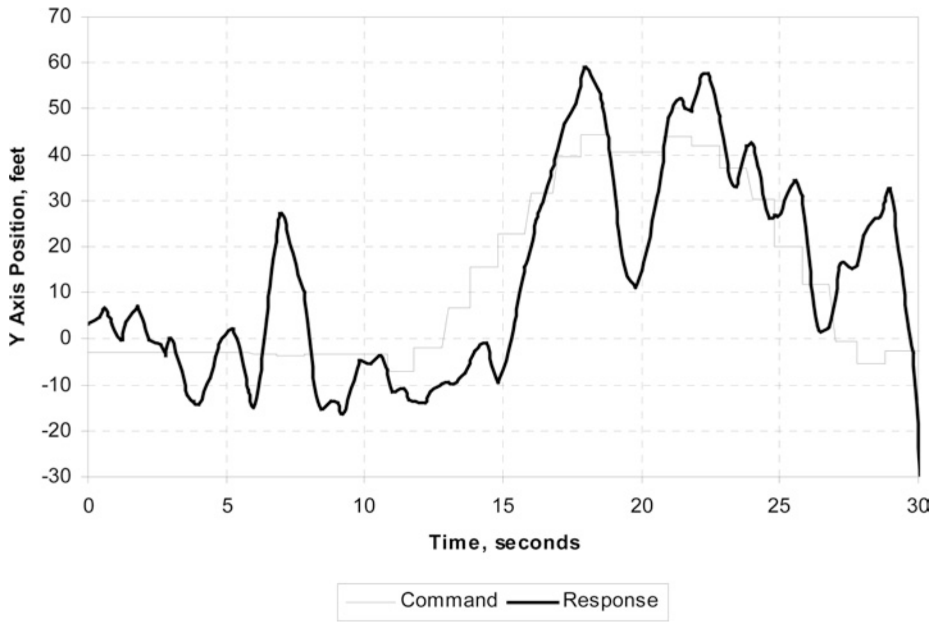

Fig. 19 Flight Test Y-Body Axis Position from Box Maneuver.

VI. Conclusions

Utilizing the FCS20 DSP/FPGA system, it was possible to demonstrate a highly capable processor combination (2 GFLOPS) together with an IMU, differential GPS, data link, video transmitter and batteries in a space slightly larger than a soda can (60 cubic inches) and weighing less than one pound. The suitability of the FCS20 for a wide range of vehicles and system requirements was demonstrated by using it for the guidance, navigation, and control of both a large GTMax rotorcraft UAV[17] and a small ducted fan UAV.[18]

This illustrates that future small UAV systems can potentially take advantage of far more complex onboard processing than is generally the case today, in this case including 16-state extended Kalman filter and a neural network adaptive flight control system. The hardware and software described could also perform image processing, obstacle avoidance, and data encryption tasks with minor hardware changes.

Future work includes further miniaturization of the overall system volume, particularly with respect to sensors and wireless communication. To effect maximum system parallelization, future plans also include moving functions currently implemented on the Nios^{TM} directly to the FPGA and subsequently eliminating the Nios^{TM}, which is currently the main data bottleneck in the system.

Acknowledgments

The authors would like to acknowledge some of the many other contributors to this work: Jeong Hur, Phillip Jones, Daniel Schrage, George Vachtsevanos, Allen Wu, and Brent Yates. This work was supported in part by the DARPA Software Enabled Control (SEC) Program under contracts #33615-98-C-1341 and #33615-99-C-1500.

References

[1]Cloudcap Technlogies website, accessed on 07/11/2005: http://www.cloudcaptech.com/

[2]MicroPilot website, accessed on 07/11/2005: http://www.micropilot.com/prod_mp2028g.htm

[3]UAV Flight Systems, Inc. website, accessed on 07/11/2005: http://www.uavflight.com/AP50.htm

[4]Athena Controls website, accessed on 07/11/2005: http://www.athenati.com/gs111m.htm

[5]Danville Signal Processing, Inc. website, accessed on 07/11/2005: http://www.danvillesignal.com/index.php?id=zx_platform

[6]Traquair website, accessed on 07/11/2005: http://www.traquair.com/catalog/microline.products.html

[7]Lyrtech Signal Processing website, accessed on 07/11/2005: http://www.lyrtech.com/DSP-development/dsp_fpga/signalwave.php

[8]Labrosse, J. J., MicroC/OS-II, "The Real-Time Kernel," CMP Books, 2003.

[9]Kannan, S. K., Koller, A. A., and Johnson, E. N., "Simulation and Development Environment for Multiple Heterogeneous UAVs," AIAA Modeling and Simulation Technology Conference, AIAA-2004-5041, Providence, Rhode Island, August 2004.

[10]Dittrich, J., and Johnson, E. N., "Multi-Sensor Navigation System for an Autonomous Helicopter," AIAA/IEEE Digital Avionics Systems Conference, 2002.

[11]Dittrich, J., and Johnson, E., "Design and Integration of an Unmanned Aerial Vehicle Navigation System," A Thesis, School of Aerospace Engineering, Georgia Institute of Technology, May 2002.

[12]Gelb, A, et al., "Applied Optimal Estimation," The M.I.T. Press, 1974.

[13]Brown, G., and Hwang, P., Introduction to Random Signals and Applied Kalman Filtering, 2nd Edition, John Wiley & Sons, 1992.

[14]Bletzacker, et al., "Kalman filter design for integration of Phase III GPS with an inertial navigation system", Computing Applications Software Technology Technical Papers, Los Alamitos, CA, 1988.

[15]Johnson, E. N., and Kannan, S. K., "Adaptive Trajectory-Based Control for Autonomous Helicopters," In the Proceedings of the AIAA Digital Avionics Systems Conference (DASC), 2002.

[16]Johnson, E., and Kannan, S., "Adaptive Trajectory Control for Autonomous Helicopters," AIAA Journal of Guidance, Control, and Dynamics, Vol. 28, No. 3, 2005.

[17]Johnson, E., Schrage, D., Prasad, J., and Vachtsevanos, G., "UAV Flight Test Programs at Georgia Tech," Proceedings of the AIAA Unmanned Unlimited Technical Conference, Workshop, and Exhibit, 2004.

[18]Christophersen, H., Pickell, W., Koller, A., Kannan, S., and Johnson, E., "Small Adaptive Flight Control Systems for UAVs using FPGA/DSP Technology," Proceedings of the AIAA Unmanned Unlimited Conference, Chicago, IL, 2004.

[19]Stevens, B., and Lewis, F., "Aircraft Control and Simulation," Wiley and Sons, 2004.