Avionics System for UAV Flight Controls Research

Subodh Bhandari*, Norali Pernalete†, Ajay Bettadapura‡, Ohanes Dadian§, Luis Andrade¶, Nigam Patel‖, Richard Lin**,

California State Polytechnic University, Pomona, CA 91768

This paper presents the research being done at Cal Poly Pomona on the development of a wireless avionics system for use on UAVs for flight controls research. The avionics system consists of an onboard flight computer for flight data acquisition and processing, and a wireless modem for live streaming of telemetry to and from the UAV to a ground station. The avionics system is being used for the implementation of guidance, navigation, and control algorithms and for hardware-in-the-loop simulation. It will also be used for the implementation of neural network based controllers that are under development. Because of its open architecture, the avionics system can be used for other research topics such as research on obstacle and collision avoidance systems for UAVs. An integrated modular avionics (IMA) architecture is being used to reduce the overall weight, size, and power consumption. A commercially available Real-Time Operating System (RTOS) is used for increased safety, security, and partitioning.

Nomenclature

AADL	Architecture Analysis & Design Language
ANN	Artificial Neural Network
ARINC	Aeronautical Radio, Incorporated
BELBIC	Brain Emotional Learning Based Intelligent Controller
COTS	Commercial off-the-shelf
DoD	Department of Defense
DURIP	Defense University Research Instrumentation Program
ESN	Echo State Network
FAA	Federal Aviation Administration
GCS	Ground Control Station
GIS	Geographic Information System
GN&C	Guidance Navigation and Control
GNSS	Global Navigation Satellite System
GPS/INS	Global Positioning System/Inertial Navigation System
GUI	Graphical User Interface
HIL	Hardware-in-the-Loop
IMA	Integrated Modular Avionics
IMU	Inertial Measurement Unit
ISR	Intelligence, Surveillance, and Reconnaissance
MLP	Multi-Layer Perceptron
MVC	Model-View-Controller
NAS	National Airspace System

*Associate Professor, Department of Aerospace Engineering and AIAA Senior Member
†Professor, Engineering Technology Department
‡Undergraduate Student, Department of Aerospace Engineering and AIAA Student Member
§Graudate Student, Department of Computer Science and AIAA Student Member
¶Undergraduate Student, Department of Aerospace Engineering and AIAA Student Member
‖Undergraduate Student, Department of Aerospace Engineering and AIAA Student Member
**Graduate Student, Department of Electrical & Computer Engineering

NDI	Nonlinear Dynamic Inversion
NSF	National Science Foundation
OCU	Operator Control Unit
PID	Proportional, Integral, and Derivative
RTOS	Real-time Operating System
S&A	Sense & Avoid
SAR	Synthetic Aperture Radar
SIL	Software-in-the-Loop
SQL	Structured Query Language
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
V&V	Verification & Validation

I. Introduction

UNMANNED Aerial Vehicles (UAVs) have become indispensable for both military and civilian use. Their military applications range from targeting drones, battlefield damage assessment and force protection to intelligence, surveillance and reconnaissance (ISR). UAVs are also used for civilian applications such as transport, scientific research, rescue missions, surveillance of fire-, flood- and earthquake-affected areas, synthetic aperture radar (SAR) interferometry, vegetation growth analysis and assessment of topographical changes. Their role is expected to increase in future years. Today, UAVs are the most dynamic growth sector of the aerospace industry.[1] Despite their potential for replacing manned aircraft, the operating cost of UAVs must be decreased while increasing autonomy, reliability, and availability. They are limited to military and restricted airspace due to safety concerns. Unmanned aircraft must be integrated into the National Airspace System (NAS) in order to realize their full potential.

NAS integration requires that the UAVs have capability to sense and avoid (S&A) obstacles and other aircraft. These capabilities help reduce the operating cost by reducing the number of crashes. The Department of Defense (DoD) has set a target of fully integrating unmanned aerial systems (UAS) into the NAS within a few years. This will allow UAVs to fly alongside manned aircraft. Due to its significance, the DoD is collaborating with NASA and the Federal Aviation Administration (FAA) to develop and mature the S&A technologies. As the role of UAVs and the level of autonomy increases, it is important to design flight controllers that perform well over the entire flight envelope.[2] Traditionally, controllers for flight vehicles are designed using linearized flight dynamics models. However, controllers based on these models only work at certain flight conditions around which the aircraft model has been linearized. UAVs are expected to fly over a wide range of the flight envelope characterized by frequent changes in dynamic pressure. Controllers based on nonlinear flight dynamic models can replace linearized controllers since they can operate over the entire flight regime. A popular method for handling parameter variations in flight control is gain scheduling.[3] The flight dynamics model is linearized at several operating points and controllers are designed for each linearized model. However, gain scheduling has many disadvantages. Frequent and rapid changes of controller gains may lead to instability, and certain flight conditions may not yield a stable linear model. Some of the disadvantages of gain scheduling can be overcome by using nonlinear controllers based on feedback linearization. One of these techniques is dynamic inversion and has been applied to UAVs and other aircraft. However, dynamic inversion requires an accurate knowledge of the aircraft flight dynamics and the controller may not work if the dynamics model is inaccurate or the system is in a non-minimum phase. Additionally, the selected controller must simultaneously handle the presence of disturbance, parameter uncertainties, modeling error, and unmodeled dynamics. Nonlinear neural-adaptive robust controllers are suitable to address these issues. Online neural networks can calculate controller parameters that take into account the changing flight dynamics and nonlinearity of the aircraft.

Today, Cal Poly Pomona is conducting research on neural network based controllers[4] and on obstacle and collision avoidance systems for UAVs.[5] For this research, a custom avionics system with an open architecture is required for the capability to have an open system that will allow modification for the addition of new hardware and software. Many commercial-off-the-shelf (COTS) avionics systems exist but are proprietary and very expensive. An open avionics suite is being designed for cost effectiveness, modularity, flight data acquisition, data processing, data telemetry, and automatic control. Designing an avionics suite is challenging

and requires selection of the sensors, processors, and form-factors that allows for a lighter, smaller, and more rugged solution. The project is funded through grant #1102382 from the National Science Foundation (NSF). Hardware for the project was purchased through a grant from the DoD under the Defense University Research Instrumentation Program (DURIP). The avionics system will be used on a number of UAV platforms at Cal Poly Pomona.

II. Background

This paper details the conceptualization, research, and development of an open avionics system for deployment on various aerial platforms at Cal Poly Pomona.

II.A. Unmanned Aerial Vehicles at Cal Poly Pomona

The Aerospace Engineering Department at Cal Poly Pomona maintains a UAV program with fixed- and rotary-wing aircraft. A few of these vehicles shown in Figures 1 and 2 include a Yamaha R-MAX helicopter donated by the Northrop Grumman Corporation, a SR-100[6] helicopter from Rotomotion, a 12 Foot Telemaster from Hobby Lobby, and multiple Sig Kadet Seniors from Sig Manufacturing. The R-MAX has a main rotor diameter of about 123 inches, an overall length of 143 inches, can handle a payload capacity of roughly 60 pounds, and is fitted with a WePilot from Viking Aerospace. The SR-100 has a main rotor diameter of 79 inches, tail rotor diameter of 34 inches and is 58 inches long, 20 inches wide, and 27 inches high. The dry weight of the helicopter is 35 pounds and the payload capacity is up to 18 pounds. It is equipped with a basic flight control system from Rotomotion for autonomous operation, telemetry gathering, and video link. The Twin-Engine is a gasoline-powered airplane with a wing span of 134 inches and is 95 inches long. It weighs 42 pounds with 25 pounds of payload capacity allowing for a large amount of fuel for long endurance missions. It utilizes the Piccolo II[7] autopilot from CloudCap for autonomous flight. The 12 Foot Telemaster, displayed in Figure 2 is 90 inches long with a wing span of 144 inches. It weighs 24 pounds with a payload capacity of roughly 12 pounds. This aircraft has also been equipped with a Piccolo II. A small fleet of Sig Kadet Seniors have also been used for multiple projects at Cal Poly Pomona. These Sig Kadets feature a wingspan of 80 inches, are 65 inches long, weigh 10 pounds empty with 6 pounds of payload capacity and are typically fitted with an ArduPilot Mega for flight controls.

Figure 1: Cal Poly Pomona UAV Fleet

Figure 2: Cal Poly Pomona 12 Foot Telemaster

II.B. An Open Avionics System for Research

The proposed avionics system is designed to meet the mission requirements of the projects and competitions at Cal Poly Pomona. These missions typically require autonomous waypoint navigation, state estimation, wireless data links, operator control units, computational power for algorithms, servo control for control surface actuation, and payload support for cameras, LIDARs, sonars, alpha-beta vanes and other equipment. The avionics system shall support an open controller architecture for changes to the Guidance, Navigation, and Control (GN&C) laws of the flight control system and the introduction of adaptive controllers and training mechanisms. Since these research algorithms and applications are currently in development, the avionics system shall also be able to isolate and tolerate faults while keeping the aircraft

operational.

Recent advances in avionics development show a trend toward an Integrated Modular Avionics (IMA) architecture,[8] which defines common processing modules, buses, and communications that provide the bandwidth to support any application. This methodology was adapted for the open avionics system at Cal Poly Pomona in order to simplify the architecture of the avionics system, and satisfy the modularity, timing, and processing requirements of the mission.

II.B.1. ARINC 653 & ARINC 664

The Aeronautical Radio, Incorporated (ARINC) 653[9] standard defines the time and space partitioning of applications running on a safety-critical real-time operating system on the common hardware. The standard also defines an APlication EXecutive (APEX) API that manages the communication, process management, timing, and monitoring of the applications running on the operating system, and interfaces to the underlying hardware. Since many of the APEX calls can be mapped to Portable Operating System Interface (POSIX) calls, a POSIX compliant operating system was desired. Low latency context switching, and preemptive thread scheduling were also required to properly isolate and task applications.

ARINC 664[10] defines an extension of Institute of Electrical and Electronics Engineers (IEEE) 802.3 Ethernet suitable for avionics communications, known as Advanced Full-Duplex Switched Ethernet (AFDX). The AFDX data network serves as the common backplane for the IMA processing modules. Since AFDX switches may not be widely available for small UAVs and student projects, standard 802.3 Ethernet switches will be used for communication. In addition to Ethernet, safety-critical applications can use redundant point-to-point RS-232, RS-429, MIL-STD 1553, or CAN bus to ensure that possible IP packet collisions and malfunctions do not affect the operation of the aircraft.

II.B.2. Size, Weight and Power (SWaP)

The size, weight and power consumption of the system must be restricted for integration on small UAVs that can only support a few pounds of payload. In that vein, the system must be able to fully support the flight control and vehicle management functions of the aircraft with minimal footprint. The design of a single module shall support control surface actuation, sensor acquisition, flight control, vehicle management and wireless data links, similar to off-the-shelf autopilots such as the ArduPilot, Piccolo, and MicroPilot. The modules should be capable of using lithium-polymer batteries that are typically used to power servos, receivers, sensors, payloads, and electric motors on the aerial platforms.

II.B.3. Algorithm Prototyping, Simulation, and Autocoding

MATLAB, Simulink and LabVIEW are typically used to design and tune controllers and algorithms. The adaptive neural network, proportional, integral, and derivative (PID) and nonlinear dynamic inversion controllers are being prototyped using MATLAB and Simulink. The API is required to interface with the Simulink Autocoder and LabVIEW Code Generator and generate inputs and outputs for the algorithms. For hardware- and software-in-the-loop simulations (HIL & SIL), FlightGear is typically used for visualization and simulation of the aircraft plant. FlightGear provides an Extensible Markup Language (XML) based input/output server that can broadcast aircraft status and orientation to the avionics system during simulation and accept flight controls input. By using Ethernet, the avionics system can communicate with FlightGear to perform HIL and SIL simulations.

III. Avionics System Architecture

The open avionics system at Cal Poly Pomona uses a set of common processing modules connected over an IEEE 802.3 Ethernet backbone as shown in Figure 3, based on the IMA architecture.[11] The modules feature a common middleware layer that supports high availability, input/output, messaging, and process management, similar to ARINC 653. The RTOS provides applications time and space partitioning for fault tolerance, isolation, and separation of criticality levels. The common processors provide computing power for adaptive flight controls, target tracking, image processing, ballistics calculations, and other applications as dictated by the mission. The data concentrator modules are designed to provide low level interfaces, signal generation, and signal conversion. This alleviates processor load on the common processing modules.

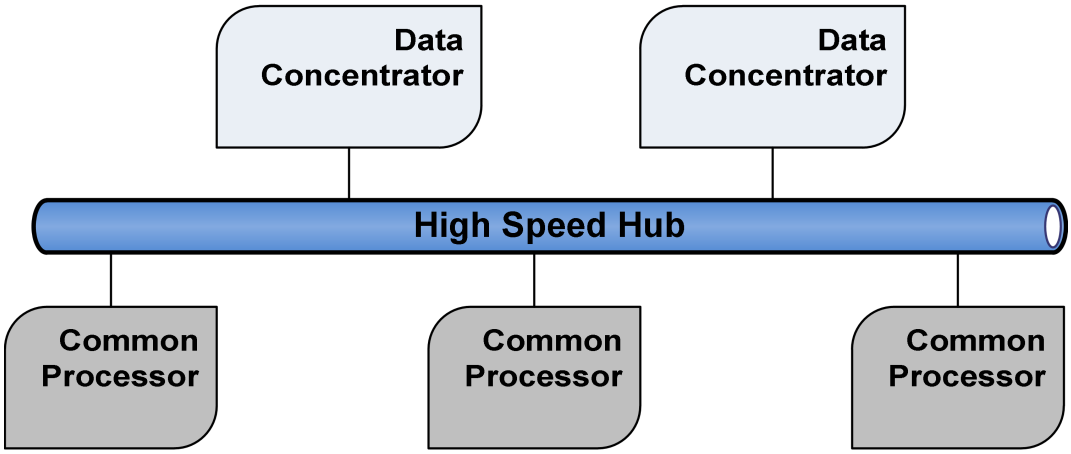

Figure 3: Open Avionics System

III.A. Data Concentrator Module

The data concentrator was designed to support the entire flight management and control system in a single unit with wireless data links to transmit aircraft status and telemetry data, since many vehicles may not support the complete integrated modular avionics system due to platform size. It is also designed to connect to a radar or radio transmit-and-receive module (TRM) analog front-end for software defined radar or radio respectively. The data concentrator board features a hybrid Field Programmable Gate Array (FPGA) and a Hard Processor Subsystem (HPS) for modularity, reconfigurability, and processing power. Analog-to-digital converters (ADC), digital-to-analog converters (DAC), random access memory (RAM), and flash memory for storage are also available onboard.

III.A.1. FPGA Subsystem

To support the input and output requirements of future payloads and aircraft subsystems, a Field Programmable Gate Array (FPGA) was selected to act as a remote input/output unit on the data concentrator board. The hardware reconfigurability and intellectual property (IP) cores of the FPGA can support various peripherals including inter-integrated circuit (I2C), transistor-transistor logic (TTL) Universal Asynchronous Receiver/Transmitter (UART), Serial Peripheral Interface (SPI), Ethernet transceivers, Universal Serial Bus (USB), MIL-STD 1553, and Controller Area Network (CAN) bus, and processing for ADC and DAC. Additionally, the discrete I/O pins on the FPGA can be used for pulse width modulation (PWM) generation and quadrature encoders for servo actuation and motor control. All of the interfaces run in parallel on the FPGA circuitry, and the IMA system accesses the data when needed. If required, a soft processor core may be loaded onto the FPGA with a Wishbone Bus or Advanced Microcontroller Bus Architecture (AMBA) to provide access to the IP cores. Real-time scheduling is provided through an RTOS that can be loaded onto a soft processor to run applications that may need direct access to the incoming data. Low level sensor filtering, image processing, radar upconversion and downconversion, Fast Fourier Transforms (FFT), and other cores can be loaded on the FPGA circuitry or soft processor as needed.

III.A.2. Hard Processor Subsystem

In addition to the FPGA, the data concentrator uses a hard processor subsystem to communicate with other common processors and data concentrators over Ethernet and USB. The hard processor system can also upload the FPGA bitstream to configure the FPGA. The common application programming interface (API) can then be used to access data from the FPGA using shared memory registers, AMBA, Wishbone, SPI, or other communication interfaces. The hard processor with a RTOS is more energy efficient than a soft core processor loaded in the FPGA fabric. If implementation of algorithms or modules for the FPGA

is time or cost prohibitive, they can be quickly developed in C or C++ on the hard processor system. The dual chip system on the FPGA also provides redundancy in the event of either an FPGA or HPS failure. Low level flight control algorithms, Kalman filtering, radar data processing, and data link processing can all be implemented on the HPS system of the data concentrator.

III.B. Common Processing Module

In order to support high bandwidth and computationally intensive applications, the common processing module provides floating point and integer performance for user applications, data processing, behavior generation, artificial intelligence, and vehicle management. The common processing module can run either a real-time operating system in a time and space partitioned environment, or on a hyper-visor with multiple operating systems based on the application requirements. This allows for isolation of non-critical applications from safety-critical real-time applications, and ensures that real-time applications are allotted enough Central Processing Unit (CPU) time to maintain aircraft operations. The common processor is connected to the avionics system through Ethernet, with USB and serial ports for additional capability. These modules may be purchased off-the-shelf, and are typically available in PC-104 and Pico-ITX form factors.

III.C. Current Implementation

The avionics system has been prototyped using a NETGEAR 5-Port Gigabit Ethernet switch as the Ethernet backplane. The AAEON PFM-CVS PC-104 form-factor Intel Atom N2600 shown in Figure 4 was used as the common processing module due to its small size and 12VDC power requirements, that can be satisfied by a 3-cell lithium polymer (LiPo) battery. The PFM-CVS has one Gigabit Ethernet port, four USB ports, and a VGA port for graphical development.

The TS-7700 board shown in Figure 5 from Technologic Systems was used as data concentrator module, and combines a Lattice XP2 8K Look-Up-Table (LUT) FPGA with a Marvell PXA166 ARM9 processor clocked at 1.066 GHz. The TS-7700 features a real-time clock for timing, dedicated Ethernet and USB ports, 55 FPGA discrete I/O, and dual redundant SD cards in the event of data corruption. A custom baseboard is in development for the TS-7700, and will eventually become a standalone design for the data concentrator. The baseboard provides breakouts and level shifting for the TS-7700 daughter board. The Aeroflex Gaisler LEON3 SPARC processor was chosen for the FPGA soft processor due to its prevalence in safety-critical aerospace applications and for its open architecture, fault tolerance, and platform independence required by the IMA system.

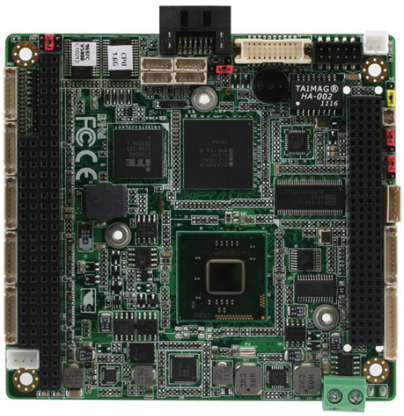

Figure 4: AAEON PFM-CVS

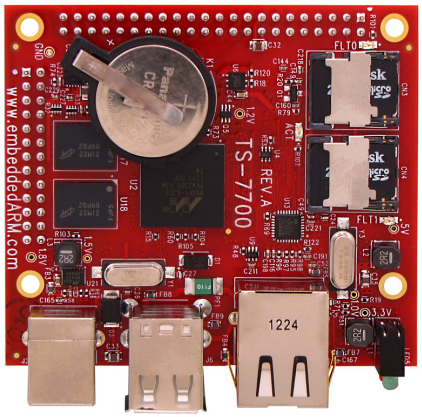

Figure 5: Technologic Systems TS-7700

For development of the adaptive controllers, a reference implementation is shown in Figure 6. This architecture was chosen in order to implement the core flight control, vehicle management, and ground control station applications for the avionics system.

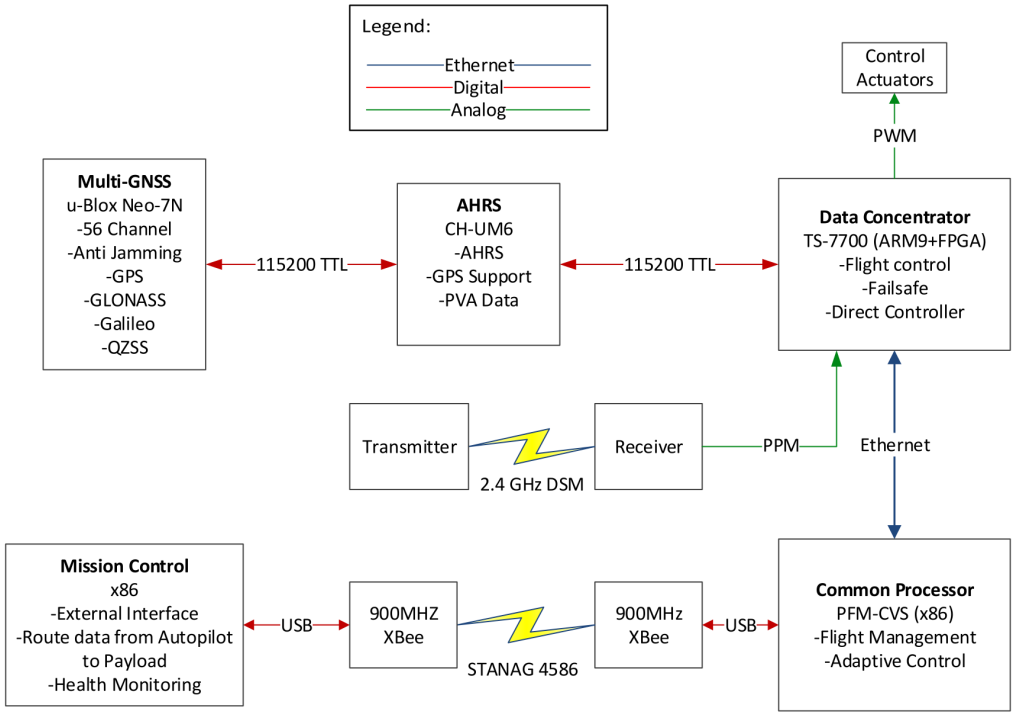

Figure 6: Prototype Reference Configuration

The Pololu CHR-UM6 Orientation Sensor shown in Figure 7 was used as an attitude heading and reference system (AHRS) that provides Kalman filtering via an integrated 32-bit ARM processor. A u-Blox NEO-7N anti-jamming Global Navigation Satellite System (GNSS) module shown in Figure 8 provides time, position and velocity (TPV) reports to the AHRS for Kalman filtering. These filtered outputs are unpacked by the data concentrator, and sent over the Ethernet bus in time-stamped quaternion format. Flight controls can be implemented on both the data concentrator and common processor for testing of adaptive flight control.

Figure 7: Pololu CH6-UMR Orientation Sensor

Figure 8: u-Blox NEO-7N Anti-Jam GNSS Module

The transmitter is a standard radio controller (R/C) that the pilot can use to manually override the autopilot in an emergency. The FPGA on the data concentrator will be used to implement a failsafe to switch between the PWM outputs of the autopilot and the pulse position modulation (PPM) of the radio receiver on the aircraft for manual override. The mission control/ground station communicates with the common processor on the avionics system using a 900 MHz XBee over a common messaging protocol. The common processor acts as the adaptive controller by running computationally intensive neural network algorithms.

The processor also runs a flight management system to provide waypoint navigation and guidance set-points for the controller on the data concentrator.

III.D. Real-Time Operating System

III.D.1. Selection Criteria

The salient features in selection of a real-time operating system were time and space partitioning of applications, message passing, fast context switching, low interrupt latency, and POSIX compliance for implementation of ARINC-653 like concepts. The RTOS shall run on both the common processor and data concentrator modules, which use x86-64 and ARM architectures respectively. It shall also support the C and C++ programming languages. Since the avionics system is intended for research and university projects, the selected operating system must provide either free non-commercial or student licensing, or full source code access. The licensing availability criterion eliminated a few commercial RTOSes from the selection, including WindRiver VxWorks, LynuxWorks LynxOS, and Green Hills Software INTEGRITY. This narrowed the selection to Micrium µC/OS-III, QNX Neutrino, FreeRTOS, embedded configurable operating system (eCos), Real-Time Executive For Multiprocessor Systems (RTEMS), and MarteOS.

III.D.2. QNX and RTEMS

Of these operating systems, QNX Neutrino was chosen due to organized documentation, the Momentics integrated development environment (IDE), and a fault tolerable microkernel architecture[12] that supports inter-process communication (IPC), distributed computing, fast interrupt latencies and context switching, process management, high availability management, and application partitioning. QNX had the best support for the target hardware, and provided a lifetime non-commercial license with unrestricted use of the development tools and operating system. RTEMS was chosen for support for the LEON3 SPARC soft processor. RTEMS and LEON3 have been widely used for a multitude of aerospace systems, including spacecraft, rockets, experimental physics, and ground installations. RTEMS and LEON3 was successfully tested on a Terasic DE0-Nano development board using an Altera Cyclone IV EP4CE22F17C6N FPGA, and is suitable for the FPGA on the data concentrator. QNX Neutrino has been installed on the PFM-CVS currently used as the common processor. The ARM HPS on the TS-7700 prototype data concentrator currently runs Debian Squeeze with a modified 2.6.34v Linux Kernel for access to the connected FPGA. This will be replaced with QNX Neutrino in the final data concentrator implementation.

III.E. Software Architecture

The software architecture in Figure 9 builds on the QNX Neutrino microkernel architecture with a common middleware featuring scheduling, process management, blackboards, message queues, protocol libraries, and a unified link API for USB, sockets and serial ports. Each avionics module will also feature a brain application that provides system reconfigurability, behavior, and planning. A health monitor controls application startup, shutdown, configuration, and fault tolerance.

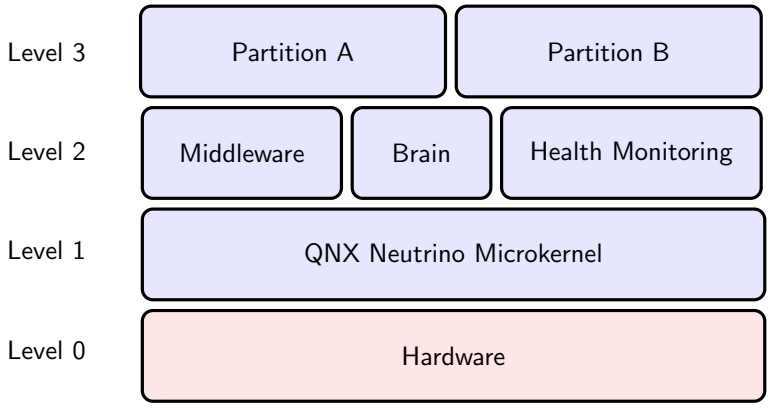

Figure 9: Software Architecture

III.E.1. Health Monitoring

The health monitoring application manages the startup and configuration of the avionics system. The implementation of the health monitor is shown in Algorithm 1. After reading a configuration file, it initializes applications and attaches them to the QNX high availability manager.[13] The high availability API defines entities that can be attached to the manager and actions the process manager can take based on the condition of the application. The QNX high availability manager also starts a guardian clone that restarts the high availability manager in the event of a crash. The health monitor performs a built-in-test (BIT) of each application before startup and reports the status of each application. The health monitoring application also attaches itself to the high availability manager to ensure it is restarted or fixed if errors occur. On restart, the monitor can perform a built-in-self-test (BIST) to ensure nominal operation. During operation, the health monitoring program can start, stop, configure, and test applications based on received messages. In the event of an error, the monitor calls an error handing function on the malfunctioning application.

Algorithm 1 Health Monitoring Thread

```
▷ Start built-in-self-test
bist()

▷ Attach self to high availability manager
self_attach()

▷ Get application configuration file
module ← get_config(ima.conf)

▷ Attach application to the high availability manager and start them
for i = 0 → num_applications do
    attach(application[i])
    start(application[i])
    i = i + 1
end for

while monitor_running do ▷ Block and handle messages when received
    msg = receive_messages()
    if msg.length > 0 then
        handle(msg)
    end if
end while
```

III.E.2. Blackboards

The ARINC 653 standard defines a blackboard sampling port similar to the operation of the User Datagram Protocol (UDP) that can be written to and read by multiple publishers and subscribers. The access methodology is shown in Figure 10. In order to provide multiple readers and writers access to a shared section, a blackboard thread is used with blocking QNX IPC MsgSend() and MsgReceive() primitives for data requests, data pushes, and client registration. The publisher threads initialize a typed named shared memory section that is passed to the blackboard thread to open. Clients register with the blackboard thread, and are given a new instance of shared memory for access. Whenever the publishers push data to the blackboard, the data is copied to an internal buffer and logged. When the client requests data from the blackboard, the client blocks, and the blackboard thread copies the internal buffer to the client's shared memory instance after the client's request message and replies after modification is finished. This ensures that access to the shared memory is synchronized and free of race conditions. Accessing the blackboard is lock-free in order to avoid deadlock, livelocks, and priority inversion. The blackboard thread blocks when idle to save CPU time. The blackboards provide typed data structures with sensor data, image processing data, system status updates and other sampled information.

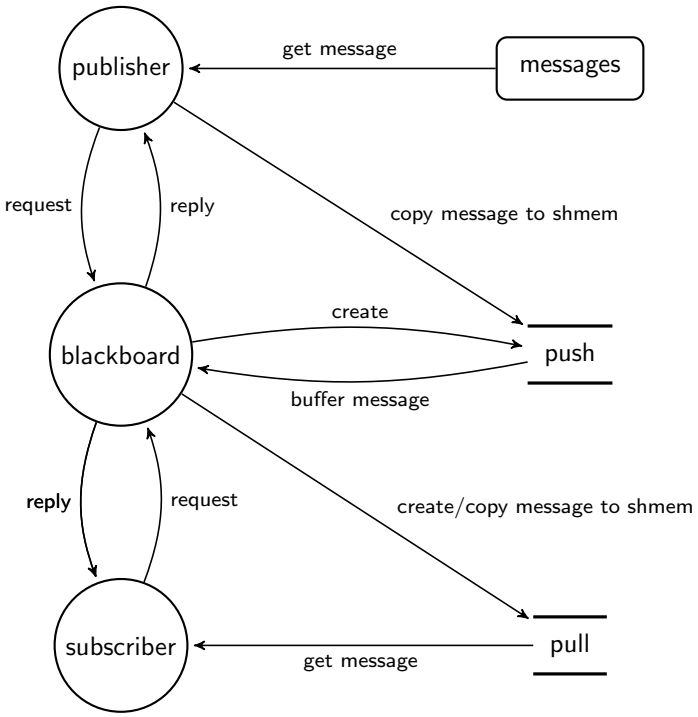

Figure 10: Blackboard Operation

III.E.3. Message Queues

In addition to blackboards, message queues are also defined in the ARINC 653 specification as queuing ports, and are provided by POSIX message queue calls. This allows for applications to queue up messages to other applications, and ensure messages are not overwritten, similar in operation to the Transmission Control Protocol (TCP). The messages are handled by the mqueue server defined in the POSIX standard. QNX Neutrino also implements an alternative queuing server called mq that uses the Neutrino microkernel architecture for higher performance through native IPC. However, this alternative server cannot use QNX Qnet for distributed messaging. Thus the selection of the messaging server is dependent on whether the applications require distributed processing or high performance messaging to components located on the same node. Currently, the message queues are used for the flight control application configuration, mode commands, and navigation and guidance set-points. These message queues are used in nonblocking mode for asynchronous delivery of data when program execution is desired regardless of whether messages are available, and synchronously when blocking is desired for program synchronization.

III.E.4. Interface API

The middleware currently defines a unified API for access to USB ports, serial ports, and network sockets. The API abstracts the specifics of the NetBSD TCP/IP stack used by QNX for network sockets, QNX io-usb driver calls, and the POSIX serial port API into open, read, write, and configuration commands. This allows interface agnostic applications to use data links based on destination locations, similar to the virtual links defined in ARINC 664 AFDX. During avionics startup, the hardware and software configuration of the avionics modules and applications are stored in configuration files, and identified by address. Applications are then provided access to sockets and serial ports based on the location of their application endpoints. If the POSIX mqueue server is enabled, and USB or serial communication is not required, applications will use message queues to send data to other applications over the network. Typically, UDP will be used for sensor data and high packet rate communications where TCP handshaking would lead to unacceptable overhead. TCP will be used for critical messages that must be guaranteed delivery. USB will be used as a redundant bus, and serial ports will be used for point-to-point communications for safety-critical systems such as flight

controls and flight data acquisition in the event of Ethernet switch failures, data collisions, and USB failures. Serial ports may be used to support legacy systems that do not support Ethernet or USB.

III.E.5. Protocol Library

The library stores a set of protocols required by the avionics system. The protocol library features data serialization and deserialization functions for packing and unpacking messages, and parsers for extracting messages from serial data streams, determining message types, and verifying checksums. In order to provide modularity and reconfigurability to the system, a variant of STANAG 4586[14] was used. This message set has been simplified and adapted for the vehicle command and interoperability requirements of Cal Poly Pomona and defines connection handling, status updates, navigation commands, mode changes, payload commands and extension messages as new capability is added. The input and output protocol of FlightGear for HIL and SIL simulations, unmanned vehicle communication protocols such as MAVLink and OpenJAUS, and interfaces to robotics platforms including the Robot Operating System (ROS), Microsoft Robotics Developer Studio (MSDR), and Urbi may also be stored in the protocol library.

III.E.6. The Brain

The brain acts as the heart of the avionics system. It is designed to maintain the integrity of the system by reconfiguring the system as the need arises. The brain must provide the capability to handle a variety of aircraft states and operating modes. It works with the health monitoring application to issue startup and shutdown commands to applications stored on the avionics system. The brain is also responsible for identifying and addressing applications within the module. After generating the function profile of the module, it broadcasts its configuration to other modules on the network, and determines the application endpoints, blackboards, message queues, and interfaces. The brain is currently under development and will use AI techniques such as neural networks, behavior trees,[15] and other automated reasoning techniques to provide natural deduction[16] to the avionics system.

III.E.7. Scheduling

The avionics system requires scheduling at both the partition and thread levels. QNX Neutrino provides an adaptive partition scheduler[17] that specifies the allocated processing bounds, criticality and free CPU time in the event of under-utilized partitions. The health monitor adjusts the partition bounds and properties based on the hosted applications and criticality levels. For example, if a flight controller is started, it will reside in a safety-critical partition and alloted CPU regardless of whether other partitions are started, in order to keep the aircraft operational. The POSIX interface provides pthread calls for managing thread scheduling. The pthreads are housed by applications that run on the adaptive partitions of Neutrino. The middleware abstracts the pthread function calls into tasks by defining configuration structures with the desired priorities, scheduling algorithms, and scheduling properties of the task. Task start and worker function pointers are used to pass functions to the threads without altering the underlying scheduling implementation defined by the middleware. This architecture allows for multiple scheduling algorithms to run on separate partitions.

Three scheduling algorithms were implemented: Earliest Deadline First (EDF), Rate Monotonic[18] (RM), and Time Triggered[19] (TT). The EDF algorithm prioritizes threads with the closest deadlines to ensure that jobs are completed and outputs are produced. The EDF can schedule threads up to 100 percent processor utilization, since the period of the thread matches the deadline and will always be scheduled to complete before the deadline. A disadvantage of the EDF algorithm is that it must accurately calculate task deadlines and store the current state of executing threads to properly schedule tasks. An extension of the time triggered scheduler known as timed multitasking was also developed and uses monitor and worker threads to trigger and produce outputs. Tasks were triggered by input conditions and outputs produced when the monitor requested the output from the worker. Between the execution of the task and outputs, other tasks could also be scheduled. This reduced the input output jitter and ensured that output is produced before requested. This allowed for deadline monitoring and task overrun handling. However, the algorithm was computationally intensive due to the use of multiple condition variables and mutex locks for triggering and signaling. A rate monotonic scheduler was then developed, due its simplicity using preemptive fixed priority assignments. Since the flight controller required multiple rate groups for sensor acquisition, algorithms and control outputs, the rate monotonic scheduler provided the most optimal scheduling priorities based on

the task frequencies. In order to avoid unbounded priority inversion while using RM scheduling, the QNX priority inheritance protocol was enabled.

III.F. Ground Control Station

The avionics system provides a ground control station (GCS) for situational awareness, avionics configuration, mission planning, data logging, and manual override of connected unmanned vehicles. The spatial-temporal data from the unmanned systems is transmitted through a datagram-based wireless connection and is presented to operator control units (OCU). The operator control units will feature flight planning, configuration, and control utilities through interactive maps with panning and zooming, drag-and-drop enabled data widgets, and tabbed interfaces. The GCS architecture is shown in Figure 11.

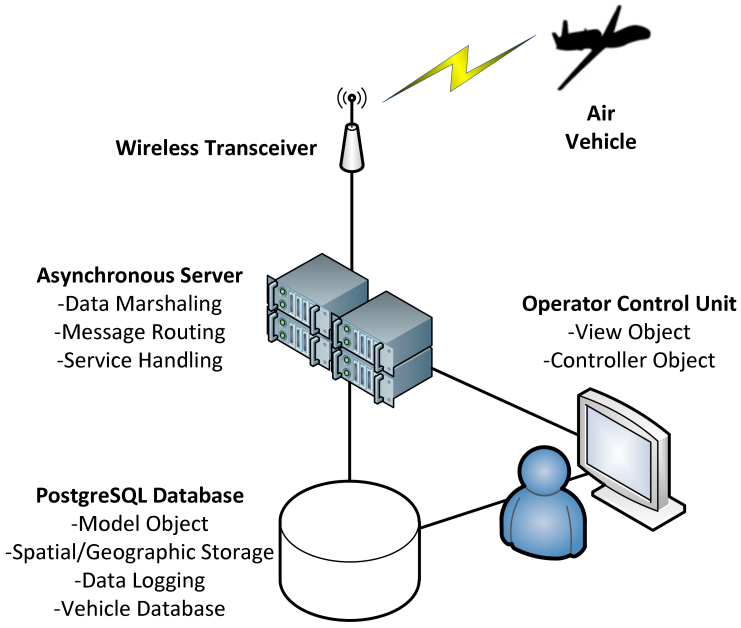

Figure 11: Ground Control Station Architecture

III.F.1. Communications

The GCS communicates with the avionics system using the variant of the STANAG 4586 protocol defined in the middleware protocol library. Messages are sent between vehicles and ground stations using XBee transceivers based on the 802.15.4 ZigBee specification. The GCS features an asynchronous server that marshals messages from attached clients and unmanned systems, and routes the packets to their destination. The server allows clients with adequate vehicle connection permissions to access the command, monitoring, and configuration services of unmanned vehicles connected to the GCS. Since the XBee transceivers are connected through point-to-point serial interfaces, the server handles requests from multiple clients that may require data from a single physical transceiver. This allows for a consistent representation of data on all client programs, as well as sanity checks by monitoring programs or the server itself on commands sent to the unmanned systems. The server provides data consistency through the Structured Query Language (SQL) by using a PostgreSQL database with a PostGIS database extender for the storage of spatial and geographic objects and other Geographic Information Systems (GIS) related data. The database stores unmanned vehicle parameters, flight plans, telemetry logs, payload data, and configuration data for the server, clients, and avionic systems.

III.F.2. Operator Control Unit

The operator control unit interface uses the model-view-controller (MVC) pattern[20] to separate presentation, logic, and data. This allows the look and feel of the application to be changed without modifying the underlying logic and data representation of the program, provided by the controller and model objects respectively. The model object can be represented by using the PostgreSQL database. This database can be manipulated by the controller based on the user interactions in the operator interface. After the controller manipulates the model, the operator control unit can query the database in order to update the view object for the user to see.

III.F.3. GUI Development

The development of the interface requires a graphical user interface (GUI) toolkit that meets certain criteria determined by the project: portability, prototyping, modularity, performance, etc. A trade study was conducted to evaluate candidates based on weighted criteria. The criteria and weights range from 0 being the least desirable to 10 being the most desirable. The candidates were C++/Qt, C++/GTK, C++/wxWidgets, C++/fltk, and C#/WinForms. The trade study displayed in Table 1 shows that Nokia's QT GUI toolkit paired with C++ is the best choice, due to its easy-to-use GUI design tool, native look-and-feel, and human-readable configuration among other benefits.

Table 1: GCS GUI Trade Study

	Weight	Gtk	Qt	wxWidgets	WinForms	OpenGL	fltk
Portability	5	4	7	9	4	6	4
GUI Tools	8	8	9	8	9	0	0
Rapid Prototyping	8	9	8	7	9	4	6
Community	10	9	7	8	7	6	5
Native Look-and-Feel	5	10	10	10	10	0	0
Modularity	7	8	8	7	10	5	7
Language Support	4	6	6	10	3	4	5
Resource-Dependence	5	8	8	8	9	5	9
Threading Support	10	7	9	7	10	7	7
System Performance	10	8	9	7	8	7	6
Documentation	10	9	9	7	10	8	7
Licensing	7	8	8	9	5	9	9
Readable Configuration	5	10	10	10	10	8	10
Knowledge Gain	8	5	5	5	5	10	5
Overall		8.02	8.27	7.87	8.16	6.01	5.85

The mapping feature will leverage Qt's WebKit framework paired with Google Maps to provide a dynamic 3D map. The connection to Google's API will be handled by QNetworkAccessManager, while the map GUI will be handled by QWebView. Since features like panning and zooming are already built-in to the API, development time will be reduced. Data will be presented using QWidgets overlayed on the map. The MVC will be implemented using Qt's model/view architecture and utilize the following classes: QAbstractItemModel for the object model, QAbstractItemView for the presentation, and QAbstractItemDelegate for the controller. Custom classes are being written to extend these abstract classes in order to provide the necessary functionality for the GCS. Communication will be handled by C++'s built-in socket library (socket.h). The application will support the Linux, Windows, and Macintosh operating systems.

IV. Flight Controls

The avionics system allows for the testing and evaluation of controllers based on various control system design techniques for UAV flight controls research. Nonlinear Dynamic Inversion (NDI), Brain

Emotional Learning Based Intelligent Controllers[21] (BELBIC), Lyapunov backstepping[22] and linear state space[23] methods are being developed for the flight control system. In addition to these controllers, neural networks are being developed for online training and controller adaptation. Model Reference Adaptive Control (MRAC) can then be used to provide a reference model for the neural networks to tune the controllers for the ideal response. These adaptive control techniques are able to overcome issues stemming from aircraft nonlinearity and plant uncertainty and are frequently used in applications of aircraft guidance and control.

IV.A. Neural Networks

Artificial Neural Networks (ANN) are a class of non-parametric models based on the neuron interactions in the human brain. ANNs can approximate virtually any function with accuracy based on the size of the available sample data, without a priori assumptions of the specific functional form. The functional form is derived directly from the example data, i.e. learning. Literature has shown that neural networks function as highly nonlinear adaptive control elements and offer distinct advantages over conventional linear controllers in terms of desired performance. There are many different classifications of neural networks. Each classification is geared toward solving certain problems such as prediction, pattern recognition, image analysis, and adaptive control.

IV.A.1. Feed-Forward

The most popular class of ANNs is the feed-forward model, whose connections do not form a directed cycle. A popular model of this class is the Multi-Layer Perceptron (MLP). This approach generally includes an input layer, single to multiple hidden layers, and an output layer.[24] In the application of adaptive learning, MLPs are coupled with an error propagation technique for online learning.

IV.A.2. Recurrent

Another class of ANNs is the recurrent neural network (RNN), which differs from feed-forward networks in that its connections form a directed cycle.[25] This correlates with the behavior of the mammalian brain. There has been much interest in using RNNs for technical applications where it is necessary to map sequences of input and output pairs, with or without a teacher. It is especially prevalent in the field of adaptive control of aerial robotics.[26] Many research efforts are averse to using RNNs due to their complex behavior, difficult implementation, and high computational requirements. However, they can have large dynamical memory and highly adaptable computational capabilities. One of the supervised learning architectures for an RNN is known as an Echo State Network (ESN), shown in Figure 12.

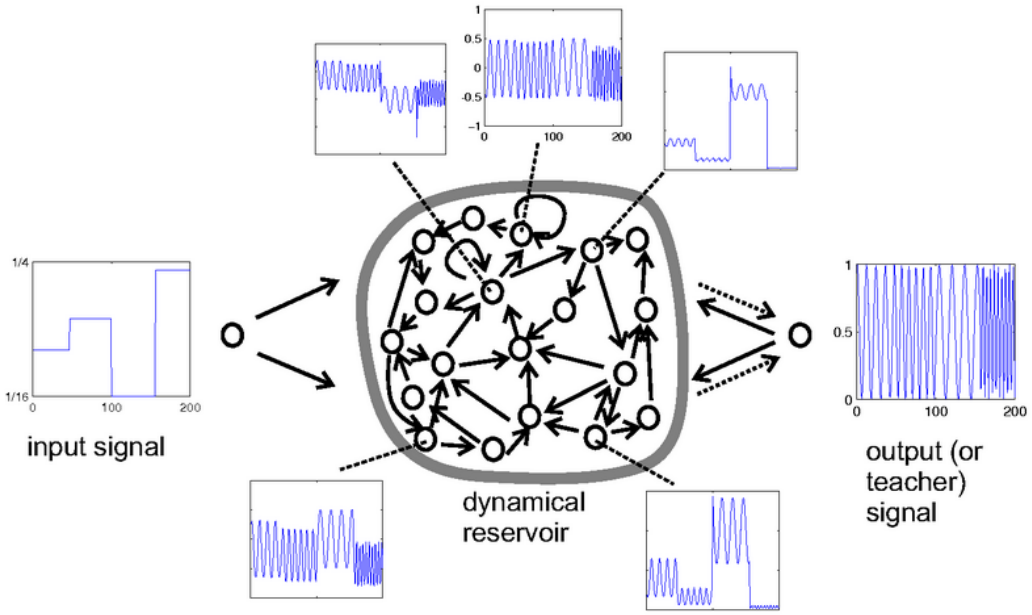

Figure 12: Echo State Network

IV.A.3. Echo State Network

The open avionics system uses an ESN to improve flight quality. This network uses a reservoir-based approach for performing nonlinear learning. Like the feed-forward model, it includes an input layer, hidden layers, and an output layer. However, the hidden layer is nonlinear and sparsely connected which allows it to solve problems for dynamical systems.[27] The echo state network activates neurons using a hyperbolic tangent function and performs Wiener-Hopf based linear regression during the read-out phase of the output layer as shown in algorithm 2. The output signals are compared against the teacher to determine if the model is fitted or if further training is necessary. The ESN has been trained using data from a Raptor 50 helicoptor and Twin-Engine FlightGear model.[24] It is being tested by a software-in-the-loop simulation comprised of a series of PID controllers, servos and state space models. The prototype was modeled in MATLAB, while the experimental model was developed in C++. The C++ implementation supports Unix-based operating systems and Windows. It is currently being integrated into to the avionics system as an application running on a QNX Neutrino partition. The matrix algebra is handled by the Eigen2.0 C++ library. The extensibility of this library allows for a well-defined hierarchy of the matrix- and vector-based behavior of signals and weight matrices in the network.

Algorithm 2 Echo State Network

```
rms = 0
▷ Generate large random reservoir
wIn = getInputMatrix;
w = getReservoirMatrix;
wOut = getOutputMatrix;
feedWM = getFeedMatrix;
yTarget = getTarget;

while rms > 0.01 do
    ▷ Compute activation states
    x = tanh(wIn[1;u(n)]+ w*x(n-1) + feedWM*y(n-1));

    ▷ Harvest the activation states
    z = [u;x];

    ▷ Build state collection matrix
    s = z^T;

    ▷ Build teacher output collection matrix
    d = yTarget;

    ▷ Compute output weights via linear regression
    y = wOut*z;

    ▷ Calculate RMSE
    mseVal = mse(y - yTarget);
    rmse = sqrt(mseVal);
end while
```

IV.B. Controller Implementation

A basic Architecture Analysis & Design Language (AADL) architecture of the flight control system is shown in Figure 13. On avionics module startup, sensor, controller, and actuator applications are tested and initialized on a safety-critical partition and attached to the high availability manager by the health monitoring application. The sensor application contains a sensor acquisition thread and a shared sensor blackboard thread. The application gathers data from the connected sensors and pushes the data into the sensor blackboard. The controller application searches for the sensor blackboard and starts controller threads that pull data from the sensor application for calculation. This application also creates a controller

blackboard that stores the generated deflections and set-points from the controller thread. Once the controller blackboard is created, an actuator module acquires the deflections, enforces deflection limits, and sends the commands to the control surfaces. The physical sensors and actuators can be connected over UDP, serial, or USB to an avionics module with an enabled flight control system. If connected over a common bus, multiple flight control systems can be started on separate modules for redundancy. Multiple sensors can be added to the common bus, and each flight control module can pull data from any of the sensors. In the case of bus failures, the flight controllers can acquire sensor data from redundant interfaces. If the flight control system is performing software-in-the-loop testing, the sensor and actuator inputs and outputs may be simulated.

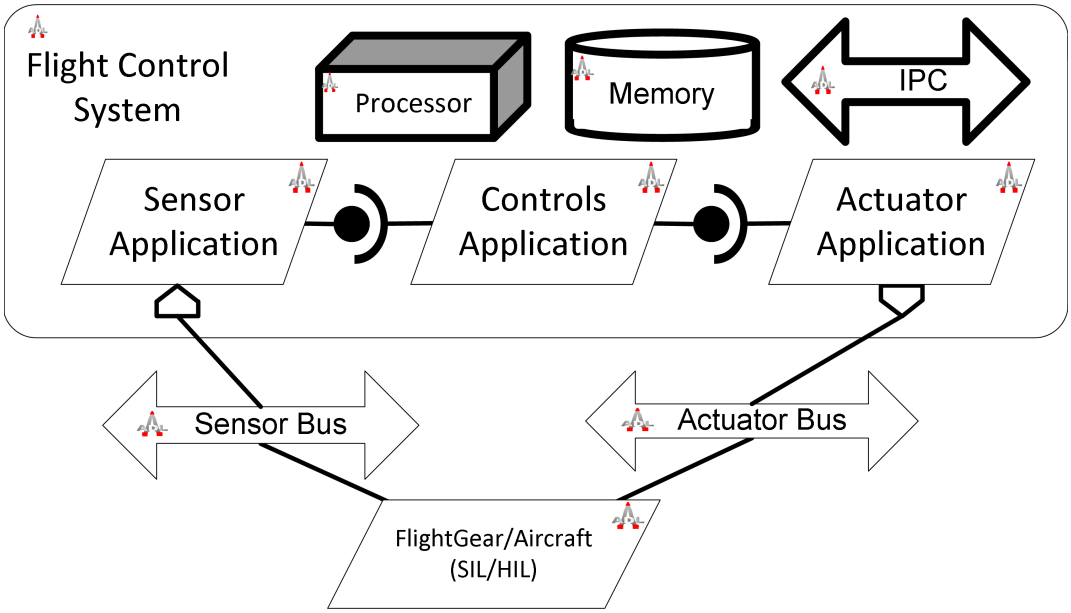

Figure 13: AADL Flight Controller Architecture

IV.C. Software-in-the-Loop Simulation

Software-in-the-loop (SIL) simulation allows for verification and validation (V&V) of the autopilot system and algorithms without risking damage to the aerial platforms. The simulation tunes the adaptive controllers, ANN and NDI, before flight tests to limit the possibility of unstable behavior due to training. The simulation tests the flight control algorithms using a FlightGear model of the Twin-Engine aircraft developed with JSBSim. The flight controller is connected to the FlightGear server in order to simulate actuators and servos. This allows for communication with the FlightGear model via UDP. A flight dynamics model in state space form of the Twin-Engine has also been developed in AVL from dynamic modeling, which is used for tuning algorithms in Simulink, MATLAB, and C++. The controller model was developed in Simulink, with a few blocks written in MATLAB and C++. Analysis is performed through MATLAB plots and loop tuning tools. The PID controllers and state space matrix were written in C while the rest of the components were written in MATLAB blocks. The motivation for the C++ implementations came from the need for more control over the algorithm. Figure 14 shows the Twin-Engine aircraft modeled in Flight Gear and controlled by the sensor, actuator, and flight control applications from a QNX Neutrino virtual machine.

Figure 14: SIL Simulation of the Twin-Engine Recovering from 20° Left Bank

The simulated Twin-Engine was sent a 20 degree left bank command. After stabilizing at 20 degrees, the roll PID channel was engaged, and the response from 20 degrees to level flight was recorded and normalized. The results are shown in Figure 15. The tuned controller demonstrated a rise time of 0.428 seconds, settling time of 1.57 seconds, and max overshoot of 5.19%. This basic PID controller is the first simulated flight control demonstration of the IMA system, and will be expanded to provide altitude hold, coordinated turn, heading hold, and velocity hold modes.

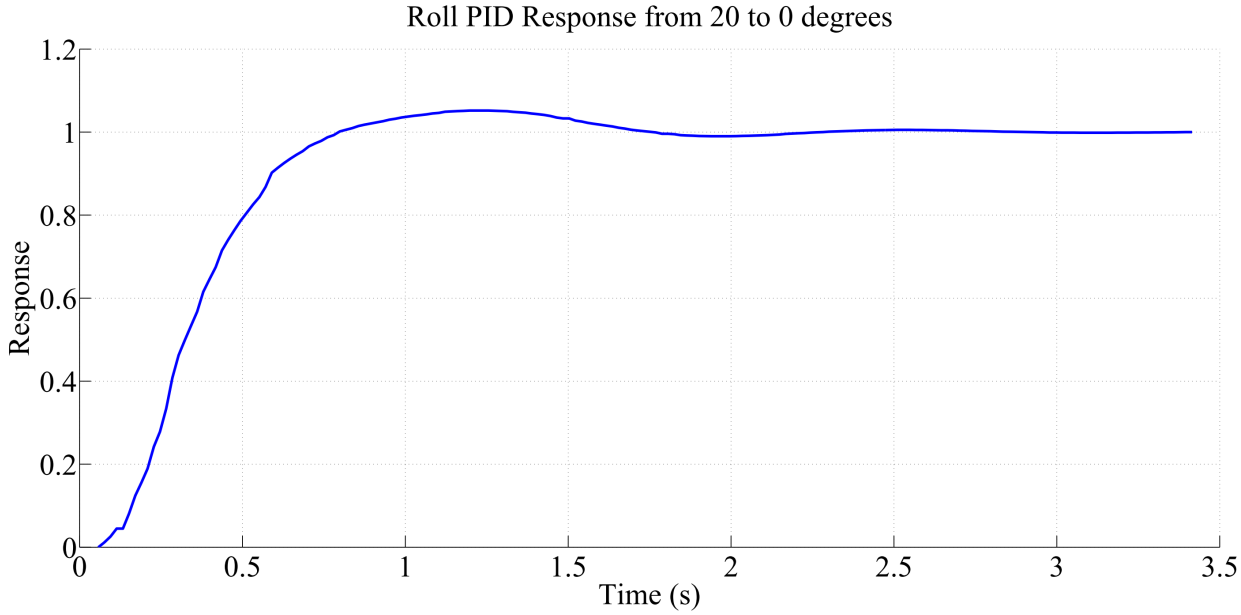

Figure 15: Closed-Loop Response of the Twin-Engine Airplane in SIL Simulation

IV.D. Hardware-in-the-Loop Simulation

Since the IMA architecture uses a common bus for both sensors and actuators, the Hardware-in-the-Loop (HIL) simulation is conducted with a configuration similar to SIL simulation. Either a common processor or data concentrator is used to host the flight control application instead of a QNX virtual machine and is connected to other modules responsible for sensor acquisition and control surface actuation. Raw sensor data is fed from FlightGear to the state estimation algorithms of the avionics system to verify the sensor fusion algorithms and Global Positioning System/Inertial Navigation System (GPS/INS) operation. During HIL, actuator operation will be checked and control surface calibration will be performed.

V. Conclusion

An open avionics system based on the IMA architecture was conceptualized and designed in order to provide the flexibility to support the needs of the various UAV projects at Cal Poly Pomona. The nascent IMA system has demonstrated initial operating capability (IOC) of health monitoring, basic flight control, blackboard ports, interface API, protocol library, data marshaling, adaptive partition scheduling, prototype neural networks in C++ and MATLAB, and the beginnings of a nonlinear dynamic inversion controller.

Although much of the reference architecture has been completed, there is still work to be done. Focus has shifted toward integration of all components with the reference hardware, controller development, hardware-in-the-loop testing, and flight testing. The GUI for operator control unit is still under development and will be integrated with the GCS server. More messages and behaviors will be also added to the system. The AI techniques for the brain application need to be trained and matured in order to avoid undefined system behaviors. HIL testing will be performed to verify and validate any additions. The IMA system will eventually support a myriad of controller techniques, payloads, and missions.

References

[1]Luley M., "UAVs around the world...an analysis of where the growth is," GE Aviation, June 13, 2012. [online], URL: http://www.theskywardblog.com/2012/06/uavs-world-%E2%80%A6-analysis-growth-3/ [cited 29, July, 2013].

[2]Holzapfel, F. and Sachs, G., "Dynamic Inversion Based Control Concept with Application to an Unmanned Aerial Vehicle," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, Rhode Island, August 16-19, 2004.

[3]Shamma, J.S. and Clutier, J.R., "Gain-Scheduled Missile Autopilot Design using a Linear Parameter-Varying Transformations," *AIAA Journal of Guidance, Control and Dynamics*, Vol. 16., No. 2, pp. 256-263, 1993.

[4]Bhandari, S., Raheja, A., et. al., "Modeling and Control of UAVs using Neural Networks," *AIAA Modeling and Simulation Technology Conference*, Minneapolis, Minnesota, August 13-16, 2012.

[5]Wagster, J., Bhandari, S., et. al., "Obstacle Avoidance System for a Quad-rotor UAV," AIAA Infotech@Aerospace 2012, Garden Grove, CA, June 19-21, 2012.

[6]Rotomotion, "Rotomotion SR100," [online], URL: http://www.rotomotion.com/r_product_4_sr100.html [cited 29, July, 2013]

[7]CloudCap Technology, "Cloud Cap Technology – Piccolo II highly integrated UAS Autopilot," [online], http://www.cloudcaptech.com/piccolo_II.shtm [cited 29, July, 2013]

[8]Alena, R. L., et. al."Communications for Integrated Modular Avionics", NASA Ames Research Center, 2006.

[9]Aeronautical Radio, Inc., "Avionics Application Software Standard Interface : ARINC Specification 653P1-3," November 15, 2010.

[10]Aeronautical Radio, Inc., "Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network: ARINC Specification 664P7-1," September, 2009.

[11]Eveleens, R., "Open Systems Integrated Modular Avionics -The Real Thing-", National Aerospace Laboratory NLR, Amsterdam, Netherlands, 2006.

[12]QNX Software Systems Limited, "QNX Neutrino RTOS System Architecture for 6.5.0," 1996-2012

[13]QNX Software Systems Limited, "QNX Neutrino RTOS High Availability Framework User's Guide for 6.5.0," 2001-2012

[14]NATO Standardization Agency (NSA), "Standard Interfaces of UAV Control System (UCS) For NATO UAV Interoperability," STANAG 4586, April 20, 2004.

[15]Grunske, L., Winter K., and Colvin, R. "Timed Behavior Trees and Their Application to Verifying Real-Time Systems," *Software Engineering Conference, Melbourne*, Vic., 2007, pp. 211-222.

[16]Robinson, J. A., Voronkov, Andrei, *Handbook of Automated Reasoning*, The MIT Press, North Holland, Vol. 2, September 1, 2001.

[17]QNX Software Systems Limited, "QNX Neutrino RTOS Adaptive Partition Scheduler for 6.5.0," 2002-2012

[18]Leung J., Kelly L., and Anderson, J.H., Handbook of Scheduling: Algorithms, Models, and Performance Analysis, CRC Press, Inc., 2004.

[19]Liu, J.,Lee E.A., "Timed Multitasking for Real-Time-Embedded Software," *IEEE Control Systems Magazine*, Vol. 23, No. 1, 2003, pp 65-75.

[20]Fowler, M., "GUI Architecture," [online], URL: http://martinfowler.com/eaaDev/uiArchs.html#ModelViewController [Cited 1 August 2013].

[21]Moren, J., Balkenius, C., "A Computational Model of Emotional Learning in the Amyglada," *Animals to Animals 6: Proceeding of the 6 International Conference on the Simulation of Adaptive Behavior*, Cambridge, Mass. 2000.

[22]Ito, H. and Mazenc, F., "Lyapunov Technique and Backstepping for Nonlinear Neutral Systems," IEEE Transaction on Automatic Control, IEEE Control Systems Society. Vol. 58, 2012, pp 512-517.

[23]Nelson, R., Flight Stability and Automatic Control, McGraw Hill, 1998.

[24]Bhandari, S., Raheja, A., "Nonlinear Control of UAVs using Neural Networks," California State Polytechnic University, Pomona, CA, June 16, 2013.

[25]Lukosevicius, M., "A Practical Guide to Echo State Networks," Jacobs University , Bremen, Germany, 2012.

[26]Ni, S., Liu, L., et. al., "Predictive Control of Vehicle Based on Echo State Network, School, Beijing Institute of Technology," *International Conference on Intelligence Science and Information Engineering*,

[27]Jaeger, H., "A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF, and the 'echo state network' approach," Franhofer Institute for Autonomous Intelligent Systems (AIS), Bremen, Germany, April 2008.