Hardware-In-Loop Simulator for Mini Aerial Vehicle

Ashish Gholkar, Amitay Isaacs and Hemendra Arya
Centre for Aerospace Systems Design and Engineering
Department of Aerospace Engineering,
IIT Bombay, Powai, Mumbai - 400076, India
ashish@casde.iitb.ac.in, {amitay,arya}@aero.iitb.ac.in

Abstract

Hardware-In-Loop-Simulator (HILS) is an important step in system design & development. In the present work a systematic approach has been adopted in creating the HILS for autonomous navigation of a Mini Aerial Vehicle (MAV). The issues addressed in this paper are 1. real time simulation of flight dynamics, 2. modeling of sensors, 3. integration of ADC and DAC interfaces under RTLinux, 4. GPS data simulator. An on-board computer and the actuators are the actual hardware in the simulation loop. The HILS is generic in nature and it can easily be adapted for any other on-board computer. This simulator will help in 1. design of navigation, guidance and control law, 2. testing the NGC laws in real time, 3. design and development of various interfaces e.g. GPS interface, filters etc. 4. identifying various faults in the system. Using the present HILS, on-board computer can be directly ported to MAV, without any modification in hardware and software.

1 Introduction

Modeling and simulation is an important step in the development of complex systems. This helps in system development and successfully shortens the design and development cycle. A good simulation is one, which can reproduce the actual behavior in virtual environment. This is only possible when a high fidelity model of the complete system is available. Many times the high fidelity equivalent is not available or difficult to model. In such cases where the system which cannot be adequately characterized by mathematical models, it is safe to embed it, as it is, into the simulation e.g. actuator dynamics is often difficult to model and the embedded microcontroller with its resident code is hard to take into account. Simulation consisting of actual components is known as Hardware-In-Loop Simulation (HILS). Early users of the HILS were from the aerospace sector, in which reliability and high performance is demanded. Presently this technology is used in every engineering field and it is also one of the methods to test the product before the actual launch. Primary requirement of HILS is real time operation and this puts lot of demand on the design of testing environment. As mentioned all the components have to perform in real-time including the simulation. Real-time means the simulation of each component is performed such that input and output signals show the same time dependent values as in real world dynamic operation.

The basic principle of HILS is that some subsystems are physically embedded within a real-time simulation model. In HILS the embedded system is fooled into thinking that it is operating with real-world inputs and outputs, in real-time. Computer software with real-time simulation capabilities and computer hardware with necessary communication abilities (A/D, D/A converters for communications with analog signals and digital ports for communication with digital signals) are necessary to perform hardware-in-the-loop simulation. In this paper such a system for autonomous Mini Aerial Vehicle (MAV) development is presented. The simulation environment is not as complex as large aircraft but still it has lot of challenges to offer while setting up such a facility in an university environment.

2 Simulation System

Mini Aerial Vehicle is quite similar to operation of a radio control aircraft and most of the hardware is common. Present simulation environment aims at enhancing the radio control aircraft capability for autonomous flying. Such small vehicle can also be

used for flight mechanics and control studies. Autonomous MAV is an aerial vehicle, which will fly through various way-points defined in space. These points can be beyond the visual range of the pilot unlike radio control aircrafts. The simulation system will help the control engineers in designing the navigation, guidance and control laws for autonomous mission.

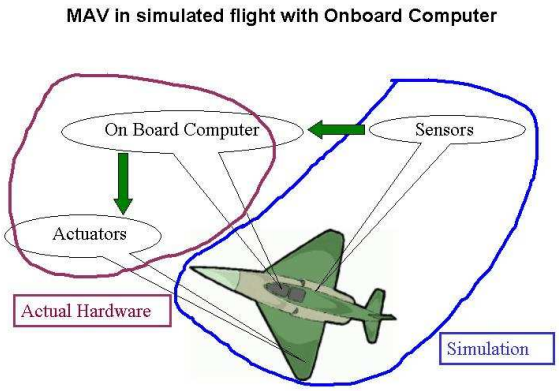

FIGURE 1: *Autonomous MAV*

During an actual autonomous mission an on-board computer will seek current state of the vehicle using various sensors and based on the NGC laws it will issue new commands to actuators to achieve the desired state. This is depicted in figure 1 above. In the simulated environment, the block on the right is virtual and the one on the left is the actual hardware.

Hardware, which can be part of such simulation system, can be radio control transmitter, receiver, on-board computer, actuators and sensors. The scheme of such a system is shown in figure 1. The steps involved in systematic development of such a simulation system are:

1. Development of full 6 degree-of-freedom (DOF) non-linear flight dynamics simulation (FDS) model
2. Verification of the FDS model
3. Development of sensor models and integrating with FDS
4. Identifying hardware for simulated sensor output
5. Integrating above and real-time simulation
6. Development of Navigation, Guidance and Control (NGC) algorithm
7. Integrating NGC with FDS, evaluate control time step by offline simulation
8. Identifying the on-board computer to execute NGC algorithm in the time step arrived by the above method
9. Integrating actuators with on-board computer
10. Integrating transmitter and receiver with on-board computer
11. Feedback from the actuators to flight dynamics simulator
12. Conduct simulation for the various NGC algorithms and missions

In the present work all except the integration of transmitter and receiver have been completed. Details about the aircraft flight mechanics and control laws will not be discussed. Emphasis is on the integration of the HILS. In the following sections details about the various systems and justification for selection is presented. FDS code is in C and executes in real time. Fourth order fixed time step Runge-Kutta method was used for time integration. Flight Dynamic Simulation is verified offline with Matlab add-on, *FDC*, authored by M.O. Rauw [1]. The (Beaver) aircraft data used in the present study is part of *FDC*. Simulation results obtained from FDS and *FDC* compared very well. The NGC algorithm uses vertical gyro for pitch and roll angle, yaw rate gyro for damping directional motion, altitude and airspeed sensors for height above ground and indicated airspeed [2]. Linear models for these sensors are used in the present work. More details about the NGC algorithm and subsystem testing are given in reference [3].

2.1 Selection of Hardware and Software

For simulation of analog sensors low cost 16-bit Digital-to-Analog Converter (DAC) card PCI-DDA 08/16 from Measurement Computing [4] is used. The PCI-DDA 08/16 is a digital to analog converter card with 8 channels at 16-bit resolution among other functional blocks. It has eight DAC channels and hence eight analog sensors can be simulated. All the channels are programmable for output range. In the present case five analog sensors are simulated.

For feedback to FDS, servos were connected to potentiometers. These potentiometers will measure the control surface deflections. 12-bit Analog-to-Digital Converter (ADC) card PCI-DAS 1002 from Measurement Computing is used. The PCI-DAS 1002 can work, in 8 channel differential mode or 16 channel independent mode (software selectable) among other functional blocks.

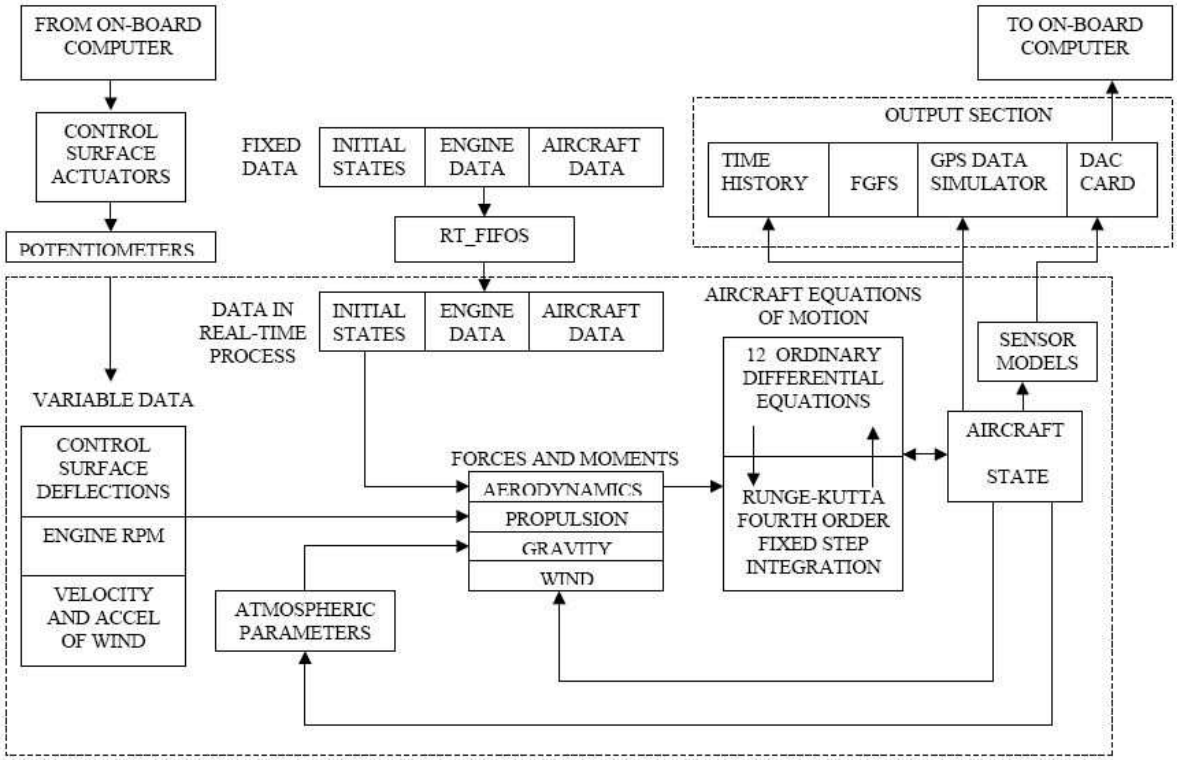

FIGURE 2: *Simulation Architecture*

Both these cards are supported by Linux Control and Measurement Device Interface (Comedi) [6] libraries. The Comedi project develops open-source drivers, tools, and libraries for data acquisition. Comedi is a collection of drivers for a variety of common data acquisition plug-in boards. It is the combination of three complementary software items:

1. a generic, device-independent API,
2. a collection of Linux kernel modules that implement this API for a wide range of cards, and
3. a Linux user space library with a developer-oriented programming interface to configure and use the cards.

The availability of open-source drivers for ADC and DAC cards was one of the primary selection criteria for using above mentioned cards.

The driver for the PCI-DAS 1002 was modified to introduce some multiplexer settling time without which there was significant coupling (cross-talk) between two adjacent analog channels. A 20 microsecond delay was introduced as multiplexer settling time by modifying driver (cb_pcidas.c) code.

Real time operating systems are expensive and many times support for the data acquisition cards is equally expensive. RTLinuxFree [5] was a natural choice. Comedi drivers provided integrated real-time support using RTLinux for most data acquisition hardware.

On-board computer selection is based on low weight, low power consumption, adequate computational power, ease of interfacing with external world etc. From literature [7, 8] it was observed that many autonomous MAV systems were using Motorola 68332 processor. Time processing units available on this processor are handy for interfacing radio control servos.

Figure 2 shows the architecture of the simulation code and how it interacts with hardware. RT-FIFOs are used to exchange data between a RT-thread and user space programs. Aircraft initial state, engine data, aerodynamic data, control surface positions etc. are initialized from files in user space and communicated to kernel space thread via a FIFO. It's easier to perform file operations in user space rather than RTLinux kernel space. Hence for storage of various time histories data is sent from RTLinux kernel thread to user space program using RT-FIFOs.

In the present FDS, 1 millisecond time step is chosen for good convergence. FDS simulation takes

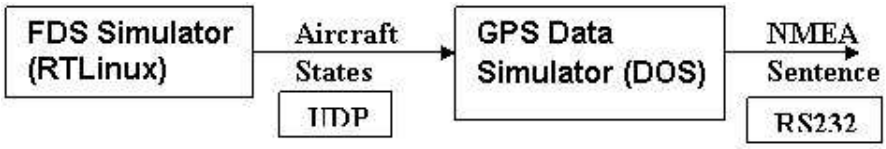

FIGURE 3: *GPS Data Simulator*

roughly 300-400 microseconds. The real time execution of each time step of 1 millisecond was also verified by sending hardware signal on digital line after completion of the each execution and it was matching within few microseconds (5-6). The digital signal was observed on 60 MHz digital oscilloscope.

2.2 GPS Data Simulator

In the present NGC, Global Positioning System (GPS) is the primary navigation sensor. Navigation information is available from GPS serially (RS232), asynchronously every one second. This information from GPS is available in the form of NMEA sentences. We use position, heading and speed over ground from the GPS data for the NGC algorithm. These values are obtained from GPGGA and GPRMC NMEA sentences.

States from the aircraft simulation (position, heading and speed) are used and appropriately converted to the GPS sentence format. This information (approx. 100 bytes) is to be given at every one second on serial port at 4800 baud. The transmission time is of the order of 200 milliseconds. Synchronizing serial communication with real time computation of the FDS is an issue. To over come this issue another computer was used for GPS data simulation. States of the aircraft were transmitted over Ethernet using UDP at the same rate as that of simulation time step (presently 1 millisecond). The size of the UDP packet is 146 bytes. In case one or two packets are lost, the data for one or two milliseconds is also lost and it will result in position error, which is insignificant compared to other errors in the actual GPS.

Schematic of GPS data simulator is shown in figure 3. UDP reception on MS-DOS platform was achieved by using a combination of TRUMPET software drivers with MS-DOS networking suite. The GPS simulator accepts UDP packets and sends GPS sentences over RS-232 to the micro-controller every one second. Initial experiments revealed that the LAN card buffered incoming network packets. In the initial implementation packets were read every one second. But due to buffering of network packets, consecutive UDP packets sent by FDS would be read every second by the GPS simulator. So instead of flight data after one second, GPS data would reflect flight data after only one millisecond. Since there was no way to clear the network buffer on-board the LAN card, some software changes had to be made on the GPS data simulator. The program sleeps till one-second expires from the time last UDP packet was read and then scans all the packets in the buffer to read correct packet corresponding to one-second latency.

As mentioned earlier, the on-board computer uses position data from the GPS and it is available as serial data and RS-232 can be really slow when it comes to real-time work. The initial approach was to use a free channel on the navigation micro-controller for incoming GPS packets over RS-232. This approach led to the micro-controller being busy for the entire duration of the GPS sentences, which at 4800 baud meant 200 milliseconds. To avoid this a dual port RAM was introduced. The GPS data was read by another micro-controller (89C52) and processed to extract only useful information to be stored in the dual port RAM. The navigation micro-controller was now free to do its work except for a few milliseconds every second when it had to read data from the dual port RAM. Mutual exclusion was implemented to avoid simultaneous read from and write to the dual-port RAM.

For graphical display, FlightGear [9] flight simulator was used. The FlightGear flight simulator project is an open-source, multi-platform, cooperative flight simulator development project available as freeware. FlightGear can accept the flight state data from an external flight simulation software over UDP. Same UDP data sent by FDS to GPS data simulator can be used by FlightGear to visualize flight.

A few advantages of having a real-time system are guaranteed worst-case jitter, high-resolution timer functions, guaranteed worst-case interrupt response times and assurance that not even a single event will be missed. Any real-time system is deterministic. Primary logic driving the RTLinux approach was timer resolution less than one millisecond and event resolution of the order of milliseconds. Developing application for real-time system is a challenging task. System crashes and reboots are very common during the development. In the present setup two-system approach was used, one for the ex-

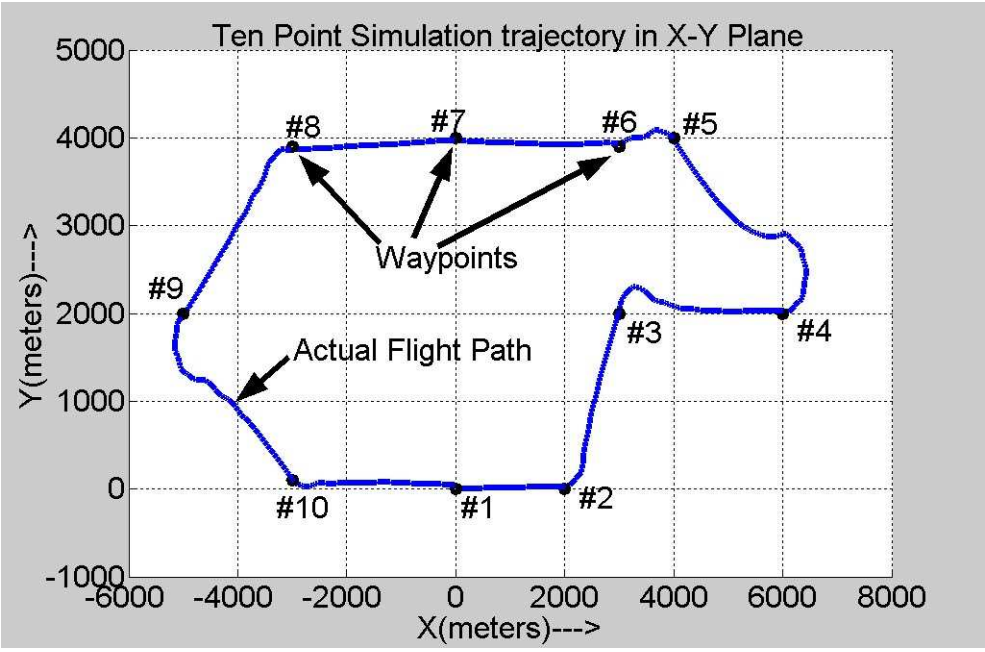

FIGURE 4: *Ten-Point Simulation Trajectory*

ecution of the real time application and other for the development of the application.

The current setup includes separate development and execution computers. The execution computer runs on RTLinux with minimal operating system environment configured with BusyBox [10]. The development computer runs on Red Hat Linux 9 and has RTLinux SDK and Comedi libraries installed. The main reason for keeping separate development and execution machines is to handle kernel panic. In the past, development and execution were on the same machine. Kernel panic caused irrevocable damage to the file system and all work was lost. By separating the development and execution processes kernel panic will not cause any data loss. This setup adds one extra computer but it is an elegant solution when frequent rebooting of the execution computer is required.

3 Results & Discussions

Complete simulation was carried out for a 10-point trajectory shown in table 1 and it is also shown in figure 4. It may be noted that the aircraft follows the specified trajectory within the given error band.

Present setup will be enhanced by incorporating the radio control transmitter and receiver and it will be possible to fly the realistic missions in which both pilot and autonomous navigation system will be in the loop.

Way-point	X Coordinate	Y Coordinate	Altitude
1	0	0	300
2	2000	0	300
3	3000	2000	300
4	6000	2000	300
5	4000	4000	300
6	3000	3900	300
7	0	4000	300
8	-3000	3900	300
9	-5000	2000	300
10	-3000	100	300
11	0	0	300

TABLE 1: *Trajectory Coordinates*

The simulator described in this paper is a good research tool for autonomous MAV NGC development. Various realistic missions will be flown before the actual flight. RTLinux and Comedi libraries are instrumental in the reduction of the development cost.

4 Acknowledgments

Authors wish to thank AR & DB for the financial support in execution of the project. Authors also wish to thank Prof. K. Sudhakar, Dept. Aero Engg, IIT Bombay for valuable suggestions in the development of HILS.

References

[1] M. O. Rauw., 2001, *A SIMULINK environment for flight dynamics and control analysis*, URL http://www.dutchroll.com.

[2] P. Srikumar and C. D. Deori, 2000, *Simulation of Mission and Navigation Functions of Nishant*, In Proc. National Workshop on Aerospace Flight Simulation, VSSC, Trivandrum, India.

[3] Vishisht V Gupta and Prasanna G. Gandhi and Hemendra Arya and Amitay Isaacs and K. Sudhakar, 2003, *Hardware-In-Loop Simulator for autonomous Navigation of Mini Aerial Vehicle*, 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2003), Singapore.

[4] URL http://www.measurementcomputing.com

[5] URL http://www.rtlinux.org

[6] URL http://www.comedi.org

[7] Marious Niculescu, 2001, *Lateral track control law for Aerosonde UAV*, 39th AIAA Aerospace Sciences Meeting and Exhibit.

[8] URL http://www.micropilot.com

[9] URL http://www.flightgear.org

[10] URL http://www.busybox.net