An Open-Source Real-Time UAS Flight Control Prototyping and Testing Platform with Fractional-Order Horizontal Controller Example*

Calvin Coopmans † Michal Podhradský Nathan V. Hoffer

Abstract

Safety and reliability are key if unmanned aerial systems (UAS) are to move from research and become an everyday part of our lives. How can safety and reliability be maintained when cost and timeliness are such pressing factors in small UAS development? In this paper, it is shown that with open-source flight control software (provided by the authors), sophisticated testing practices (hardware-in-the-loop) can provide rapid pre-flight verification of experimental control system designs. Using these methods increases the reliability of flight control software developed in many ways if tested properly. Both PID and auto-generated $PI^{\lambda}D$ control schemes are compared as examples of flight software under test, and simulated flight performance is shown.

1. INTRODUCTION

Unmanned aerial systems (UAS) are experiencing exponential growth for civilian applications. There are several issues which prevent UAS from being fully utilized as an everyday tool. The major obstacle to be solved is how to integrate UAS safely into the civil airspace. In this context, safety means obstacle avoidance, communication between manned and unmanned aircraft, and robust and fault tolerant systems [1],[2], & [3]. This work is based on previously published work showing the use of hardware-in-the-loop (HITL) for model verification [4].

Since 2006, AggieAir has been developing unmanned aerial vehicles and analytical tools and processes in pursuit of water-related scientific remote sensing. Through a service center at the Utah Water Research Laboratory (UWRL) at Utah State University, AggieAir utilizes both vertical takeoff and landing (VTOL) and Fixed-wing platforms to research the collection of important data about crop health, stream habitat, invasive species, etc. Data collection is the mission of AggieAir, and many system-level factors affect the quality of the data missions such as safety, reliability, stability, etc. The AggieAir UAS platforms are small, low-cost, autonomous, with multispectral remote sensing capabilities [5]; a current AggieAir fixed-wing UAS platform is shown in Fig. 1, and a current hexarotor VTOL platform is shown in Fig. 2. For flight control, the open-source Paparazzi autopilot system [6] is used for all platforms, with low-cost electronics and navigation sensors, while maintaining excellent flight characteristics and reliability [7].

1.1. Challenges of Flight Code Testing

All program code must be tested and verified, and aerospace code is no exception–perhaps the furthest from one. One of the main advantages of small UAS is the low-cost and short design cycle times in comparison to manned aircraft. Thus, constrained tightly during design and testing, small UAS are faced with unique software testing challenges for upholding airspace safety, as well as system resilience and mission performance in varied conditions. Model checking [8] and other techniques such as code standards can and should be used if possible–for example, DO-178B/C describes the safety standards of airworthy software [9], where IEEE 29119 is a testing standard rubric which can be applied more generally [10]–as well as formal function methods in design (for example, SpecTRM-RL [11]).

For research purposes, it is desirable to show an implementation of various arbitrary controllers or other publishable flight code, which adds the problems of auto-generated code to the list: higher difficulty for humans to audit code increases the likelihood of an unintended behavior. While no amount of testing will reveal all code defects, a basic level of "stability" must be assured before any code is put into flight, and therefore after generation, all flight code requires testing and verification no matter its source.

1.2. Motivation

Pushing the boundaries of what is possible with UAVs while maintaining a high level of safety is challenging. Carefully designed tests are crucial for evaluating that a design (such as a control system) is reliable enough for flight. Testing in the airspace has risks and the ability to run tests in a controlled environment such as a virtualized one is invaluable for rapid creation and implementation of new small UAS elements such as control and navigation algorithms, etc. Since data collection for scientific information is the

*This work was supported by Utah MLF 2006-2016 with the AggieAir group at Utah State University (http://aggieair.usu.edu/)

†c.r.coopmans@ieee.org

mission of AggieAir, improvements in these system design elements can provide gains in data quality which can be crucial in enabling new applications and better scientific outcomes.

An industry standard for demonstrating the functionality of flight control systems is hardware-in-the-loop (HITL). This is a testing scheme during which flight hardware and software are applied to a numerical model of an aircraft while the flight hardware and software respectively run as though in a real flight. All flight systems can also be tested in-the-loop, and edge cases such as hardware/software failure can be applied to a designed system in a safe and controlled environment, leading to better overall robustness and reliability, i.e. safety.

1.3. Contribution

The main contribution of this work is to show how HITL provides a tool for testing and rapid implementation of flight code while maintaining a high level of safety and reliability. Designers can proceed even with low-cost tools such as an open-source UAS autopilot – Real-Time Paparazzi (RT-Paparazzi), and when executed properly, testing and verification techniques such as HITL can be used to show the robustness of the system in varying circumstances and thus derive the benefits of advanced testing. Even small, low-cost UAS such as the AggieAir Ark platform used in this work can benefit from aerospace testing rubrics, which contribute to greater reliability and higher levels of safety for all entities in the airspace.

Also, a working example of the RT-Paparazzi HITL is given to reinforce its usefulness. The example contains a comparison of a standard Real-Time Paparazzi PID controller, Simulink auto-generated PID controller, and a Simulink auto-generated $PI^{\lambda}D$ factional-order controller.

It is the author's intent to make the HITL source code freely available for any user of the Paparazzi community, to enable more developers to write more fully tested, safer autopilot code. The work associated with this publication will be made available as part of the RT-Paparazzi project after final publication at `http://wiki.paparazziuav.org/wiki/RT_Paparazzi`

2. Real-Time Paparazzi and Hardware-in-the-Loop as a research platform

The following section gives an overview of the real-time Paparazzi project and the HITL system. The challenges and advantages of using auto-generated code are outlined. The section concludes with a brief description on how the real-time Paparazzi hardware-in-the-loop system has been tested and verified and shows that it is ready for the testing of experimental flight code.

Figure 1: AggieAir Fixedwing Platform – Minion, prepared for takeoff

Figure 2: AggieAir Multirotor Platform – Ark, ready for takeoff

2.1. Real-Time Paparazzi Unmanned Aerial System

Paparazzi [6] is an open-source autopilot. Its main advantage over similar systems (such as Pixhawk or ArduoPilot) is high modularity and configurability–it is easy to customize for needs of the user. An additional feature of Paparazzi is the real-time autopilot code which AggieAir has contributed heavily to the development of. The original "non-real-time" Paparazzi is running on a microcontroller without an operating system. This so-called "bare metal" architecture runs all periodic tasks and events together in an infinite loop, which results in a simpler and smaller code (which might be important when the memory is scarce), but the timing depends solely on the cooperation of the tasks. In other words, if any of the tasks within the loop takes longer than anticipated (for example a blocking read on a serial port or a computation-heavy calculation), the other tasks will be delayed because they have to wait for this task to finish. On the other hand, using a real-time operating system (RTOS) allows the tasks to be distributed into separate threads, where each thread has its memory and priority. That way if a lower-priority task (such as path planning routine) takes longer

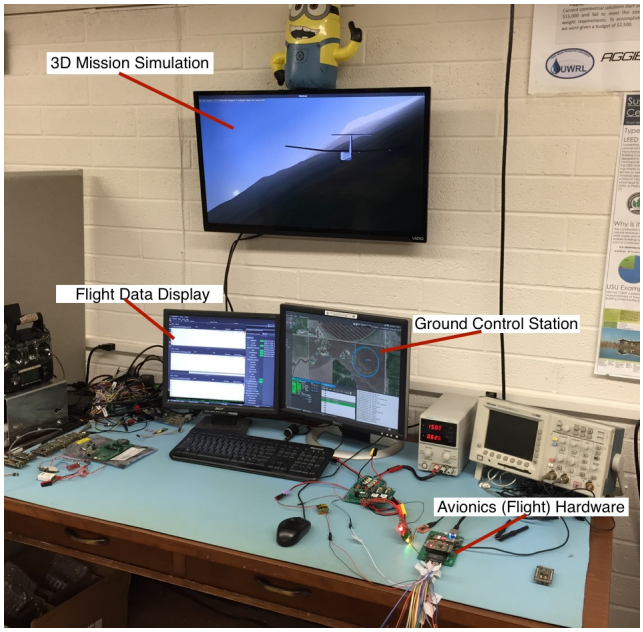

Figure 3: HITL test station in simulated flight

than usually, other higher priority tasks will still execute in a timely manner, thanks to the internal OS scheduler. As a result, the real-time aspect of the autopilot ensures consistent timing and increased reliability of the autopilot code. The underlying RTOS also monitors resource allocation, such as CPU load and free memory.

Paparazzi, similarly to other open source autopilots, uses PID control loops for attitude and altitude stabilization as well as navigation control (path following). Although more advanced control approaches have been successfully used on UAVs, PID control is still the most common in the world of small UAVs. PID controllers provide sufficiently good flight performance and have been proven over many flight hours.

2.2. Hardware-in-the-loop simulation

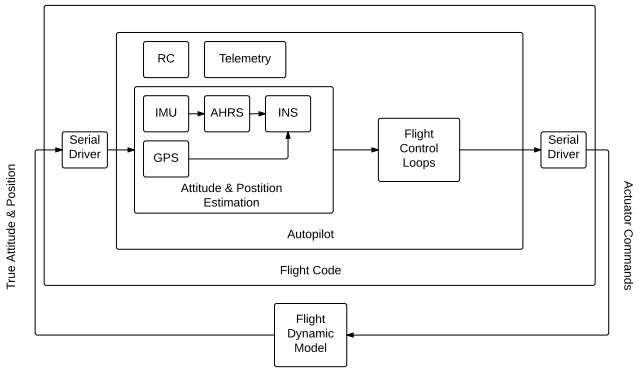

Figure 4: Hardware-in-the-loop (HITL) autopilot testing block diagram

Hardware-In-The-Loop (Fig. 4) runs actual flight code on the autopilot hardware (flight code and flight hardware), simulating the navigation sensor inputs (in this case serial packets) and aircraft flight dynamics model (FDM) – the physical response of the aircraft to control inputs and disturbances. Hardware-in-the-loop is the simulation closest to real flight because both the hardware and the code are identical to the set being used in real flight – the autopilot is really flying with artificial sensor data.

Indeed, a challenge is to have a precise FDM, because the closer the FDM is to the real aircraft, the closer HITL simulation is to real flight. It is even possible to simulate changing battery voltage (based on the power used by actuators), airspeed etc. using either replacements for digital sensors or having DAC converters providing the desired value, guided by the simulator. In this paper, we use a nominal, simplified FDM to demonstrate the effects of simulation on differing controllers, so the importance of the FDM is lessened for this example.

Complementary to HITL is another method of testing, called Software-In-The-Loop (SITL). SITL runs the flight code in a simulated environment on a host machine and doesn't require the flight hardware to be used. The benefit of SITL is that it is simpler to set up (no need for additional hardware), but it is less close to the real flight since the flight code is compiled for a different platform (typically x86 or x64) and run on a desktop computer, instead of a microcontroller.

In the ideal case, the flight performance in SITL and HITL would be identical, but that is often not the case for several reasons. First, the timing in HITL is very important– the autopilot has to receive the right data at the right moment, and the FDM has to receive control inputs from the autopilot at the right time. In SITL, there is more leeway, since the timing is only pseudo real-time (SITL can "catch up" or "slow down" the simulation as needed). Second, the code in HITL includes drivers, hardware abstraction layer (HAL) and other control-unrelated code which might cause slightly different responses (rounding errors on 32bit vs 64bit platforms, for example).

2.3. Challenges and Advantages of Auto-Generated Flight Code

Many researchers desire to demonstrate novel new controllers and control methods. For the majority of these research efforts, the experimental prototyping tool of choice is The Mathworks' MATLAB, which on its own contains an impressive amount of generally reliable code. With all the code in The Mathworks' MATLAB Central file exchange (`http://www.mathworks.com/matlabcentral/`), the amount of accessible code for nearly any application is tantalizing. Then, with C-code generation tools (MATLAB Coder), it is possible to create C-code which will compile and run in many environments such as a UAS autopilot. This is nearly never done because the risks of flying such computer-produced code are very grave–without oversight,

any code taken from an unknown source (MATLAB Central) and converted into a less readable language (C) is foolish at best.

Thus, the central struggle of rapid prototyping's code accessibility and the need to uphold safety and resilience during actual flights is clear. In order to fly safely, we must protect flight code from all of the following (among others [12]):

- Any unplanned resource usage, such as RAM and CPU usage.
- Memory leaking, unexpected memory access without a memory management unit.
- Unplanned numerical behavior in controllers (instability, poor performance).
- Unexpected code execution (stemming from stack poisoning, etc. as well as memory access bugs in the code).

Only proper design and planning will grant the best chances of crash avoidance, but in case that rapid results are needed given specific safety guidelines, it can be possible to avoid the most prevalent bugs and establish a baseline performance before human-reviewed flight code can be produced. By fully simulating auto-generated code in HITL, and applying as many edge cases as possible while carefully monitoring the resource usage with e.g. RT-Paparazzi, it can be shown that a particular controller or other flight critical system performs in all of the cases planned for by the designers. While this can never show absolutely all of the errors in any code beyond basic complexity, for small UAS it is an invaluable tool to develop and prove any new controller, including automatically generated controllers.

2.4. Previous AggieAir Research on HITL Validation

This paper builds on earlier work showing the use of HITL for model verification [4]. This work validated HITL simulation with real flight data, using the actual flight control code and flight plans.

These "flights", shown in Fig. 5 were remarkably similar (within 10% error, despite a windy flight day), leading to conclusion that with an accurate flight dynamics model, the HITL is a sufficient representation of real flight and can be used for rapid development and tuning of control loops, as well as testing of new autopilot extensions (i.e. additional navigation routines or intelligence).

3. Example Real-Time Paparazzi HITL Software Test: Fractional-Order Control

To demonstrate the capabilities of the Real-Time Paparazzi (RT-PPRZ) HITL system, it is desired to replace some of the VTOL control loops with Simulink Coder auto-generated code, and then to implement a novel controller

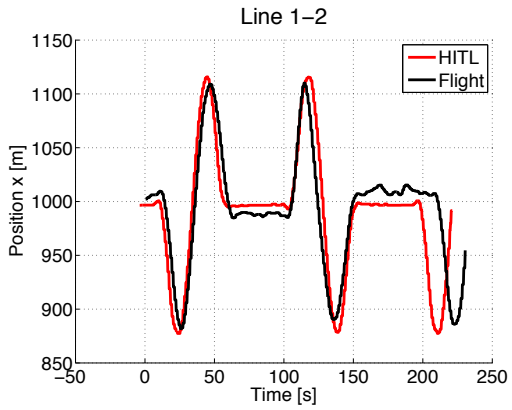

Figure 5: Line 1-2 test results: X-position vs time of HITL and Minion flight data.

in Simulink which can be tested in HITL and shown to be functional before performing actual flight tests. In this case, the horizontal position (X, Y) controller is replaced by a Simulink equivalent; its performance is then shown to be equivalent to the RT-PPRZ native controller with the same topology. Then a modified version (a fractional-order integrator feedback controller) is generated and used to show that the Simulink auto-generated code is also viable for flight testing, and that results are as expected. Finally, the CPU and static memory usage for each control scheme are compared to show that they are all reasonable for the given autopilot hardware. Note that RT-Paparazzi's altitude (Z) controller is unchanged so if any of the experimental controllers malfunction the VTOL craft will not crash and control can be regained easily by the safety pilot. A simulated wind was added to the HITL simulations to excite the dynamics of the controllers more clearly and provide a closer-to-real-world example.

3.1. PID Horizontal Position Controller

The RT-Paparazzi position controller scheme is shown in Fig. 6. This controller follows a waypoint in space predefined in the flight plan and includes PID elements as well as a feed-forward. This controller is a typical PID with saturation and integration limiting, allowing for simple, reliable, general flights in most wind conditions.

3.1.1. RT-Paparazzi horizontal PID controller. The baseline for the RT-Paparazzi position controller is regular flight – that is CPU and RAM usage during a nominal flight with unmodified controllers in HITL. This can be found in Fig. 9, and represents a typical flight.

3.1.2. Simulink auto-generated horizontal PID controller. The original RT-PPRZ horizontal controller was recreated in Simulink, and integrated into the RT-PPRZ autopilot, replacing the PID controller above. After some conversion to and from the RT-PPRZ units, the control perfor-

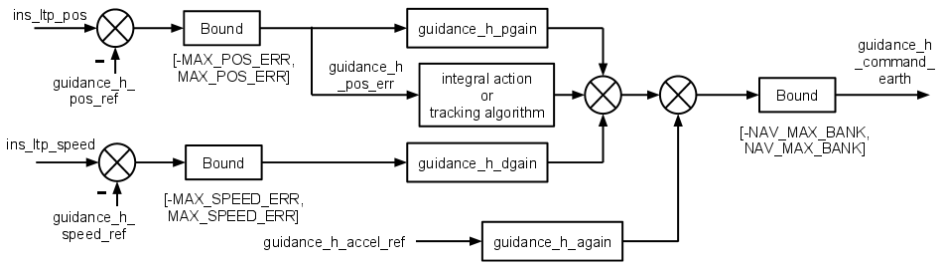

Figure 6: RT-Paparazzi VTOL horizontal controller diagram from [6]

mance of this controller is identical to the original RT-PPRZ controller (Fig. 9), while the CPU usage is similar and RAM usage is slightly higher, indicating a less efficient method of performing the same simple control task. Figure 7 shows the Simulink project that implements the RT-PPRZ controller.

3.2. PI$^\lambda$D Factional-Order Controller

To demonstrate the ability of the RT-PPRZ HITL platform to perform a research task, a novel controller was chosen. Fractional order calculus and control (FOC) is an interesting topic in controls research [13] and has been approached by other researchers doing small UAS research in the past, leading to favorable results vs. complexity of inputs [14, 15, 16]. Due to the techniques of the discretization (discussed below), the FOC controller implementation uses a constant, finite amount of numerical processing resources and memory. Thus, the complexity of the FOC discretization can be chosen to budget CPU and RAM usage on the autopilot with reliable results.

3.3. Fractional-Order Calculus

Fractional calculus is a generalization of integration and differentiation to non-integer order fundamental operator ${}_aD_t^\alpha$, where a and t are the limits of the operation. The continuous integro-differential operator is defined as

$$ {}_aD_t^\alpha = \begin{cases} \frac{d^\alpha}{dt^\alpha} & : \alpha > 0, \\ 1 & : \alpha = 0, \\ \int_a^t (d\tau)^{-\alpha} & : \alpha < 0. \end{cases} \tag{1} $$

The two definitions used for the general fractional differintegral are the Grunwald-Letnikov (GL) definition and the Riemann-Liouville (RL) definition [17, 18]. The GL is given here

$$ {}_aD_t^\alpha f(t) = \lim_{h\to 0} h^{-\alpha} \sum_{j=0}^{\left[\frac{t-a}{h}\right]} (-1)^j \binom{\alpha}{j} f(t-jh), \tag{2} $$

where $[.]$ means the integer part. The RL definition is given as

$$ {}_aD_t^\alpha f(t) = \frac{1}{\Gamma(n-\alpha)} \frac{d^n}{dt^n} \int_a^t \frac{f(\tau)}{(t-\tau)^{\alpha-n+1}} d\tau, \tag{3} $$

for $(n-1 < \alpha < n)$ and where $\Gamma(.)$ is the *Gamma* function.

The Laplace transform method is used for solving engineering problems. The formula for the Laplace transform of the RL fractional derivative (Eq. 3) has the form [17]:

$$ \int_0^\infty e^{-st} {}_0D_t^\alpha f(t)\, dt = s^\alpha F(s) - \sum_{k=0}^{n-1} s^k \; {}_0D_t^{\alpha-k-1} f(t)\Big|_{t=0}, \tag{4} $$

for $(n-1 < \alpha \leq n)$, where $s \equiv j\omega$ denotes the Laplace operator.

Some other important properties of the fractional derivatives and integrals can be found in many existing works (e.g., Oldham and Spanier [18], Podlubny [17], and others).

3.4. Fractional-Order Filtering

In order to implement fractional-order calculus in a real-world system, a practical approximation must be made. While it is possible to realize fractional-order elements in analog hardware – "passive" hardware devices for fractional-order integrator, such as fractances (e.g., RC transmission line circuit and Domino ladder network) [19] and Fractors [20], there exist some restrictions, since these devices are currently difficult to tune. An alternative feasible way to implement fractional-order operators and controllers is to use finite-dimensional integer-order transfer functions or fractional-order signal processing.

The fractional-order differentiator and fractional-order integrator are fundamental building blocks for fractional-order signal processing. The transfer function of fractional-order integrator (FOI) is simply

$$ G_{FOI}(s) = \frac{1}{s^\alpha} \tag{5} $$

where α is a positive real number. Without loss of generality, we only consider $0 < \alpha < 1$. The transfer function of fractional-order differentiator (FOD) is simply

$$ G_{FOD}(s) = \frac{1}{G_{FOI}(s)} = s^\alpha. \tag{6} $$

In time domain, the impulse response of $G_{FOI}(s)$ is

$$ h(t) = L^{-1} G_{FOI}(s) = \frac{t^{\alpha-1}}{\Gamma(\alpha)}, \quad t \geq 0. \tag{7} $$

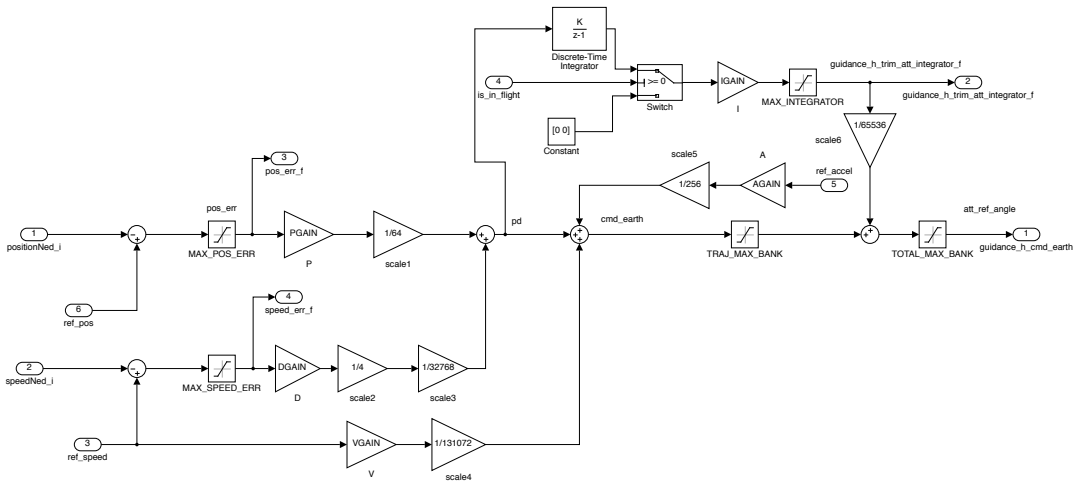

Figure 7: Simulink implementation of RT-PPRZ horizontal guidance controller with feedforward

Replacing α with $-\alpha$ will give the impulse response of fractional differentiator s^{α}.

A FOI or FOD is an infinite-dimensional system. So, when implemented digitally, it must be approximated with a finite-dimensional discrete transfer function. This is the so-called "discretization" problem of FOI or FOD [21]. The reader is referred to Chen et al.[22], Monje et al. [23], and Krishna [24] for excellent reviews and tutorials on discretization issues.

As stated, an integer-order transfer function representation of a fractional-order operator s^{α} is infinite-dimensional. However, it should be pointed out that a band-limited implementation of a fractional-order controller (FOC) is important in practice, i.e., the finite-dimensional approximation of the FOC should be done in a proper range of frequencies of practical interest. In this chapter, the recursive frequency-space discretization techniques of Oustaloup et al. [25] are used. Interestingly, the fractional-order α could even be a complex number.

3.4.1. PPRZ PID vs Auto-generated Fractional PID autopilot. A small research goal was set: attempt to employ fractional order feedback control in the RT-PPRZ horizontal waypoint controller. To accomplish numerical simulations of FOCFs, the FOMCON [26] toolbox was used–a full toolbox for simulation of fractional-order implementations in MATLAB Simulink, including discretized blocks ideal for embedded code generation. The modified RT-PPRZ fractional-order Simulink system used is seen in Fig. 8. In this case, the discretization (order 20) limits memory usage and, coupled with the low order of $\alpha = 0.8$, is assumed to make initial conditions for the integrator trivial during the flight time. It should be noted that gains and other parameters were chosen somewhat by trial and error, however unscientific this might be. For this test, it is clear that the model does fly, and the Simulink code does compute in deterministic time, allowing the normal autopilot functions to work properly and succeed in the task of the research goal. As seen in Fig. 9, the CPU usage is actually similar, and still not beyond a generally safe limit for real-time systems (75% CPU usage).

3.4.2. HITL Comparison Test Results. To summarize the control simulation results: Figure 9 shows that the estimated CPU time of the three controllers is almost identical, while the Simulink code increases the static RAM usage as expected. The CPU (a 168Mhz STM32F4) is bound in I/O most of the time and added numerical load did not change the estimated usage. The fractional-order integrator was not scientifically tuned for these tests (although this is planned for a future publication), but clearly yielded better tracking results in the 6 m/s simulated wind of the simulation as seen in Figs. 10, 11, and 12. Overall, the auto-generated code flew the craft in HITL successfully and did not use up significantly more resources than the non-auto generated code. Many hours of testing would be required before actual flight testing of this code, and it is possible with HITL to do this kind of endurance testing (i.e. an order of magnatude greater than the longest expected mission of any AggieAir craft).

4. CONCLUSIONS

The main contribution of this paper is in demonstrating the usability of hardware-in-the-loop (HITL) testing for low-cost, open-source, small civilian UASs. In particular, HITL is well suited for testing flight code on the ground in a controlled and safe environment. The usability of HITL is shown through the example RT-PPRZ HITL software test which compares simulation results of 3 controllers: PPRZ standard PID, Simulink auto-generated PID, and Simulink auto-generated fractional-order PI$^{\lambda}$D controller.

The comparison shows that each of the arbitrarily tuned controllers have similar CPU usage while the fractional order controller has significantly better performance position

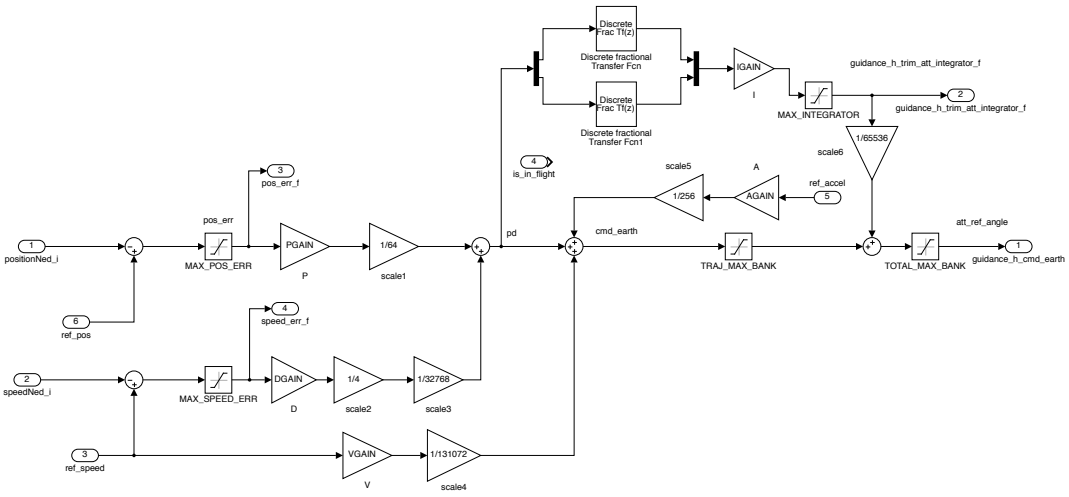

Figure 8: Simulink implementation of RT-PPRZ fractional horizontal guidance controller with feedforward

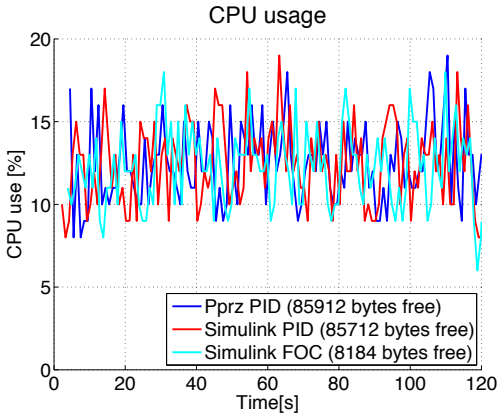

Figure 9: Summary of the three control implementations and their estimated CPU usage during testing

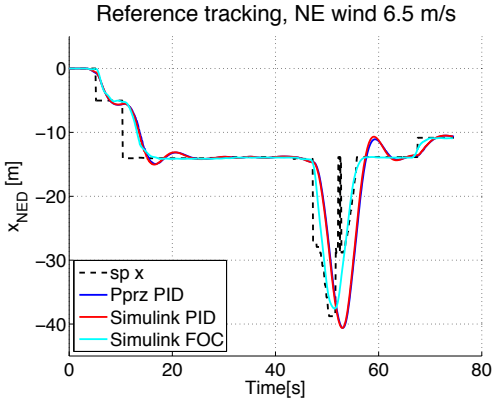

Figure 10: X-tracking of the three controllers in 6 m/s simulated wind

following in simulated wind.

The AggieAir group and the authors believe in Open Source software, and are proudly making the code for RT-PPRZ and HITL simulation publicly available, so UAV developers and researchers can verify their control systems before a flight, thus increasing safety and reliability of UAV operation.

Future work will include actual flight testing of the FO controller code with real-world (windy) data as well as new Simulink controllers.

ACKNOWLEDGMENT

The authors would like to acknowledge Dr. Mac McKee of the Utah Water Research Laboratory for his support. This work is supported by Utah Water Research Laboratory MLF 2006-2016. The authors would also like to thank the members of AggieAir who made it possible to make this paper and the Paparazzi UAV community for providing the wonderful Paparazzi project.

References

[1] Federal Aviation Administration, "Integration of civil unmanned aircraft systems (UAS) in the national airspace system (NAS) roadmap," FAA, Tech. Rep., 2013.

[2] K. Dalamagkidis, K. P. Valavanis, and L. Piegl, "On unmanned aircraft systems issues, challenges and operational restrictions preventing integration into the national airspace system," *Progress in Aerospace Sciences*, vol. 44, no. 7-8, pp. 503–519, oct 2008.

[3] K. Dalamagkidis, K. P. Valavanis, and L. A. Piegl, "Current status and future perspectives for unmanned aircraft system operations in the US," *Journal of Intelligent and Robotic Systems*, vol. 52, no. 2, pp. 313–329, feb 2008.

[4] C. Coopmans, M. Podhradský, and N. V. Hoffer, "Software- and Hardware-in-the-Loop Verification of Flight Dynamics Model and Flight Control Simulation of a Fixed-Wing Unmanned Aerial Vehicle," in *Workshop on Research, Educa-*

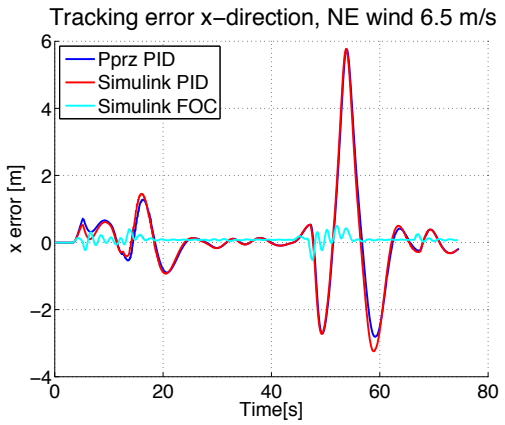

Figure 11: X-error terms of the three controllers in 6 m/s simulated wind

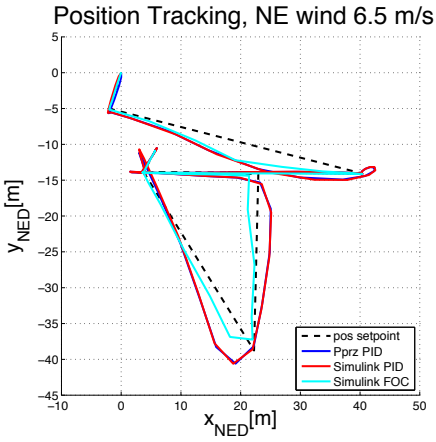

Figure 12: X- and Y-position tracking of the three controller implementations with simulated wind

tion and Development of Unmanned Aerial Systems, 2015.

[5] A. M. Jensen, Y. Chen, M. McKee, T. Hardy, and S. L. Barfuss, "Aggieair - a low-cost autonomous multispectral remote sensing platform: new developments and applications," in *Proceedings of the 2009 IEEE International Geoscience and Remote Sensing Symposium*, vol. 4, jul 2009, pp. IV–995–IV–998.

[6] Paparazzi Forum, "Open-source Paparazzi UAV project." [Online]. Available: http://paparazzi.enac.fr/

[7] C. Coopmans, B. Stark, and C. M. Coffin, "A payload verification and management framework for small UAV-based personal remote sensing systems," in *Proceedings of the 2012 Int. Symposium on Resilient Control Systems (ISRCS2012)*. IEEE, 2012, pp. 184–189.

[8] P. R. Gluck and G. J. Holzmann, "Using SPIN model checking for flight software verification," in *Aerospace Conference Proceedings, 2002. IEEE*, vol. 1, 2002, pp. 1–105–1–113 vol.1.

[9] L. A. Johnson, "Do-178B, software considerations in airborne systems and equipment certification," *Crosstalk, October*, 1998.

[10] S. Kawaguchi, "Trial of organizing software test strategy via software test perspectives," in *Proceedings of the 7th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2014, p. 360.

[11] "Safeware engineering corporation: SpecTRM features." [Online]. Available: http://www.safeware-eng.com/softwaresafetyproducts/features.htm

[12] M. J. Karlesky, W. I. Bereza, and C. B. Erickson, "Effective Test Driven Development for Embedded Software," in *Electro/information Technology, 2006 IEEE International Conference on*, may 2006, pp. 382–387.

[13] C. A. Monje, Y. Chen, B. M. Vinagre, D. Xue, and V. Feliu-Batlle, *Fractional-order systems and controls: fundamentals and applications*. Springer Science & Business Media, 2010.

[14] Y. Luo, H. Chao, L. Di, and Y. Q. Chen, "Lateral directional fractional order (PI) alpha control of a small fixed-wing unmanned aerial vehicles: controller designs and flight tests," *Control Theory Applications, IET*, vol. 5, no. 18, pp. 2156–2167, dec 2011.

[15] W. Chunyang, L. Jinfei, L. Yiduo, and L. Mingqiu, "Research on UAV attitude control based on fractional order proportional integral controllers," in *Control Conference (CCC), 2014 33rd Chinese*, jul 2014, pp. 3552–3557.

[16] J. Han, L. Di, C. Coopmans, and Y. Chen, "Pitch Loop Control of a VTOL UAV Using Fractional Order Controller," *Journal of Intelligent & Robotic Systems*, vol. 73, no. 1-4, pp. 187–195, 2014.

[17] I. Podlubny, *Fractional differential equations*. Academic press San Diego, 1999.

[18] K. B. Oldham and J. Spanier, *The Fractional Calculus*. Dover, 1974, vol. 17.

[19] I. Podlubny, I. Petráš, P. O'Leary, L. Dorčák, and B. M. Vinagre, "Analogue realizations of fractional order controllers," *Nonlinear dynamics*, vol. 29, no. 1, pp. 281–296, 2002.

[20] G. Bohannan, "Analog realization of a fractional control element-revisited," in *Proceedings of the 41st IEEE International Conference on Decision and Control, Tutorial Workshop*, vol. 1, no. 1, 2002, pp. 27–30.

[21] Y. Chen and K. L. Moore, "Discretization schemes for fractional order differentiators and integrators," *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, vol. 49, no. 3, pp. 363–367, 2002.

[22] Y. Chen, B. M. Vinagre, and I. Podlubny, "Continued fraction expansion approaches to discretizing fractional order derivatives - an expository review," *Nonlinear Dynamics*, vol. 38, no. 1-4, pp. 155–170, dec 2004.

[23] C. A. Monje, Y. Chen, B. Vinagre, D. Xue, and V. Feliu, *Fractional Order Systems and Control - Fundamentals and Applications*. Springer-Verlag, 2010.

[24] B. T. Krishna, "Studies on fractional order differentiators and integrators: a survey," *Signal Processing*, vol. 91, pp. 386–426, 2011.

[25] A. Oustaloup, F. Levron, B. Mathieu, and F. M. Nanot, "Frequency-band complex noninteger differenciator: characterization and synthesis," *IEEE Transactions on Circuits and Systems, I: Fundamental Theory and Applications*, vol. 47, no. 1, pp. 25–39, 2000.

[26] A. Tepljakov, E. Petlenkov, and J. Belikov, "FOMCON: Fractional-order modeling and control toolbox for MATLAB," *Proc. 18th Int Mixed Design of Integrated Circuits and Systems (MIXDES) Conference*, pp. 684–689, 2011.