Robustness Through Simplicity: A Minimalist Gateway to Neurorobotic Flight

Simon D. Levy
Washington and Lee University

Abstract

In attempting to build neurorobotic systems based on flying animals, engineers have come to rely on existing firmware and simulation tools designed for miniature aerial vehicles (MAVs). Although they provide a valuable platform for the collection of data for Deep Learning and related AI approaches, such tools are deliberately designed to be general (supporting air, ground, and water vehicles) and feature-rich. The sheer amount of code required to support such broad capabilities can make it a daunting task to adapt these tools to building neurorobotic systems for flight. In this paper we present a complementary pair of simple, object-oriented software tools (multirotor flight-control firmware and simulation platform), each consisting of a core of a few thousand lines lines of C++ code, that we offer as a candidate solution to this challenge. By providing a minimalist application programming interface (API) for sensors and PID controllers, our software tools make it relatively painless to for engineers to prototype neuromorphic approaches to MAV sensing and navigation. We conclude our discussion by presenting a simple PID controller we built using the popular Nengo neural simulator in conjunction with our flight-simulation platform.

1 Introduction

Beginning with J.J. Gibson's pioneering research on visual perception (Gibson, 1979), decades of research in behavioral neuroscience have shown the importance of robust, tightly-coupled perception/action cycles in supporting successful movement (predation, obstacle avoidance) in challenging environments. This is especially true for flying animals like birds and insects, whose survival depends on overcoming of a variety of forces in three-dimensional space; most obviously, gravity (Floreano, Zufferey, Srinivasan, & Ellington, 2009).

In attempting to build neurorobotic systems based on flying animals, engineers have come to rely on existing firmware and simulation tools designed for miniature aerial vehicles (MAVs). Although they provide a valuable platform for quick entrée into the world of FPV racing or aerial photography (firmware), and the the collection of data for Deep Learning and related AI approaches (simulation), such tools are deliberately designed to be as feature-rich and general as possible, to appeal to the widest audience. The most popular software tools support air, ground, and water vehicles and provide a hierarchy of safety mechanisms for minimizing the likelihood of injury and property damage. Unsurprisingly, the sheer amount of code required to support such broad capabilities can make it a daunting task to adapt these tools to building neurorobotic systems for flight.

In the remainder of this paper we present a pair of simple, object-oriented software tools – *Hackflight* and *MulticopterSim* – each consisting of a core of a few thousand lines of C++ code, that we offer as a candidate solution to this challenge. These software tools are built on the popular Arduino microcontroller platform and the popular video game platform Unreal Engine 4. By providing a minimalist application programming interface (API) for sensors and PID controllers, these tools make it relatively painless to for engineers to prototype neuromorphic approaches to MAV sensing and navigation.

2 Hackflight

Hackflight began in 2015 as an attempt by the author to build a simple open-source flight-control firmware

Appearing in Proceedings of the Workshop on Robust Artificial Intelligence for Neurorobotics (RAI-NR) 2019, University of Edinburgh, Edinburgh, United Kingdom. Copyright 2019 by the authors.

program for MAVs using the Arduino platform (Banzi & Shiloh, 2014). At that time, as well as today, there were two major firmware projects for MAVs: ArduPilot (ArduPilot Dev Team, 2019a) and Cleanflight (Cleanflight Team, 2019). ArduPilot focuses on sophisticated mission planning with waypoint navigation and other features, and runs mainly on the Pixhawk flight controller. Cleanflight and its derivatives (Betaflight, Raceflight) are popular with FPV (first-person-view) racing enthusiasts, and run on a broad variety of flight-control boards designed for FPV racing. (A more recent Cleanflight derivative, iNav, adds features for navigation and for fixed-wing aircraft). Although both projects can trace their origin to the Arduino platform, they have long since switched to using their own non-Arduino hardware drivers for sensing and motor control. Both projects are supported by large development teams and have a code base of several hundred thousand to two million lines (see Table 1). Hackflight, by contrast, uses approximately 4,000 lines.[1]

How can Hackflight get away with using to or three orders of magnitude less code than the two most popular flight-control firmware packages? We attribute this difference to a few important design principles.

2.1 Features

Unlike ArduPilot, which supports a variety of vehicle types (multirotors, fixed-wing aircraft, ground vehicles, marine vehicles), Hackflight supports only multirotors. Cleanflight and its derivatives, while supporting mainly multirotors (and perhaps fixed-wing aircraft), offer a variety of configuration features and flight modes (PID controllers), allowing everyone from beginners to professional racing pilots to use them. Hackflight, by contrast, uses only a the bare minimum of PID controllers necessary for stable flight, allowing you to create your own PID controllers with relative ease (see section 2.4 below).

2.2 Audience

Although both ArduPilot and Cleanflight are open-source, their target users are mostly non-programmers. There is therefore a heavy focus in both projects on GUI-based configurator programs. Hackflight, by contrast, is targeted toward engineers and researchers comfortable with coding in C++. Adding a feature to Hackflight therefore requires significantly less code support, enabling rapid prototyping of new sensors, PID, controllers, etc.

[1]To estimate the number of lines of code in each package, we cloned the package repository from github, ran the **cloc** program (https://github.com/AlDanial/cloc) in the root directory of the repository, and summed over the reported number of lines in C/C++ header files, C files, and C++ files.

2.3 Arduino compatibility

As mentioned above, Hackflight began as the author's attempt to to build a simple open-source flight-control program using the Arduino software libraries. Although Hackflight now supports a subset of the STM32F3/4 flight controllers supported by Cleanflight and its derivatives, our focus has always been on Arduino compatibility. Thanks to the recent availability of small, fast, 32-bit microcontroller development boards like Teensy and the STM32L4 line from Tlera Corporation[2], Arduino compatibility is no longer tied to slower, eight-bit boards lacking floating-point support (see Figure 1). Arduino compatibility means that Hackflight can quickly exploit the increasing variety of new sensors available today, without the need to write a custom driver. Although the variety of neuromorphic sensors currently available cannot rival the variety of Arduino-compatible MEMS sensors (inertial measurement units, proximity sensors, and the like), we are optimistic that neuromorphic devices will follow the same trajectory; i.e., they will provide a UART or other low-level serial interface for working with Arduino and similar development platforms.

2.4 Simple object-oriented API

Hackflight is written entirely in C++, with the core components written in header-only style. Our focus is on object-oriented design, with most classes (altitude PID control, distance sensing) being subclasses of other, more abstract classes (PID controller, sensor). In addition to enabling extensive code re-use, this approach allows us to abstract the driver code for a component (sensor, motor) from the algorithms using that component (Madgwick quaternion filter, mixer). This clean separation allows Hackflight to be "dropped" directly into a simulation environment (through the use of C++ `#include` statements), without the need for Hardware-In-the-Loop" (HIL), socket connections, or other indirect mechanisms (see section 3 below). Although both ArduPilot and Cleanflight separate the driver code from the algorithmic code, Hackflight's consistent use of object-oriented design allows us to avoid pre-processor macros (`#ifdef .. #else .. #endif`) that are used extensively in those two packages and can make it difficult to arrive at a basic understanding of much of the code.

As well as keeping the codebase small, simple, and portable, these design principles support a more di-

[2]https://www.tindie.com/stores/TleraCorp/

rect connection between the mathematical theory underlying flight control and its implementation in code. Figure 2 illustrates this point by showing the main loop in Hackflight. In the figure, each box (demands, state) represents a simple datatype in the C++ code, and each oval (R/C Receiver, Sensors, PID controllers, Mixer) represents an abstract class. Mathematically, then, each abstract class is a function from one datatype to another: $\mathsf{Sensor} : \mathsf{State} \mapsto \mathsf{State}$; $\mathsf{PIDController} : (\mathsf{State} \times \mathsf{Demands}) \mapsto \mathsf{Demands}$. We believe that this design principle makes Hackflight both easy to understand and simple to adapt.

Figure 3 illustrates these principles by showing a complete Arduino firmware sketch (main program) for a quadcopter using the flight controller in Figure 1. As the sketch shows, Hackflight's simple API supports programs in which only the required components (PID controllers, receiver, mixer) need to be specified (as opposed to choosing from a list of options with a control statement). This approach results in example programs that are easy to read and to adapt for use with new sensors, vehicle designs and control paradigms.

Table 1: Approximate size of flight-control firmware packages

Package	Lines of code
ArduPilot	2,310,889
Cleanflight	851,659
Hackflight	4,269

Figure 1: Arduino-compatible flight controller for Hackflight (total cost approx. $55 U.S.)

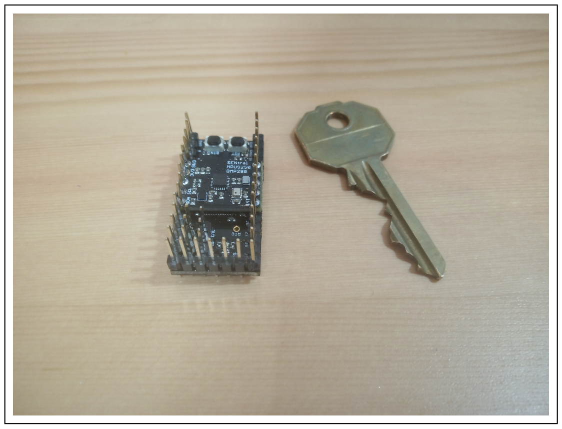


Figure 2: Hackflight main loop

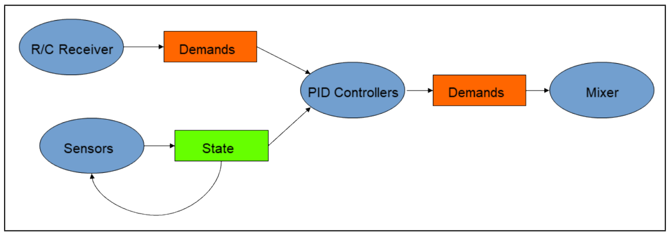
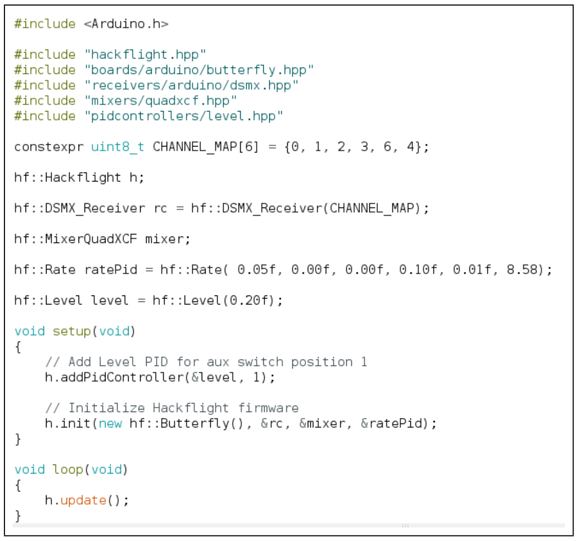


Figure 3: Sample Hackflight sketch for Arduino



```
#include <Arduino.h>

#include "hackflight.hpp"
#include "boards/arduino/butterfly.hpp"
#include "receivers/arduino/dsmx.hpp"
#include "mixers/quadxcf.hpp"
#include "pidcontrollers/level.hpp"

constexpr uint8_t CHANNEL_MAP[6] = {0, 1, 2, 3, 6, 4};

hf::Hackflight h;

hf::DSMX_Receiver rc = hf::DSMX_Receiver(CHANNEL_MAP);

hf::MixerQuadXCF mixer;

hf::Rate ratePid = hf::Rate( 0.05f, 0.00f, 0.00f, 0.10f, 0.01f, 8.58);

hf::Level level = hf::Level(0.20f);

void setup(void)
{
    // Add Level PID for aux switch position 1
    h.addPidController(&level, 1);

    // Initialize Hackflight firmware
    h.init(new hf::Butterfly(), &rc, &mixer, &ratePid);
}

void loop(void)
{
    h.update();
}
```

3 MulticopterSim

MulticopterSim began as a plugin for the popular V-REP robotic simulation platform (E. Rohmer, 2013). In March of 2017 the head of Microsoft's AirSim project (Shah, Dey, Lovett, & Kapoor, 2017) contacted the author about using Hackflight as the flight-control software for AirSim, a new quadcopter flight simulator built on the UnrealEngine4 platform (Sanders, 2016), citing design principles of Hackflight is the primary reason for this interest. After a licensing incompatibility ended up making this collaboration unfeasible, the author turned to developing quadcopter flight simulator from scratch, using UE4 and and the Hackflight firmware.

In addition to its focus on Deep Learning, AirSim has since expanded to include support for self-driving cars, and provides Python APIs for remote operation of the vehicles. As with flight-control firmware discussed in the previous section, this rich set of features translates into significantly more code. Table 2 shows the relative sizes of AirSim and MulticopterSim, based on the same metric used in Table 1. As we saw with Hackflight, the design principles used in MulticopterSim help keep the codebase small, manageable, and easily extendable.

The core of MulticopterSim is the abstract C++ *FlightManager* class. This class provides support for running the vehicle dynamics and the PID control regime (e.g., Hackflight) on its own thread, after it first disables the built-in physics in UE4. The dynamics we used are based directly on the model presented in (Bouabdallah, Murrieri, & Siegwart, 2004), written as a standalone, header-only C++ class that can be easily adapted for other simulators and applications if desired. This class also supports different frame configurations (quadcopter, hexacopter) via virtual methods. By running the FlightManager on its own thread, we are able to achieve arbitrarily fast updates of the dynamics and flight-control. We currently limit the update rate to 1kHz, based on the data output rate of current MEMS gyrometers. It would also be possible to run the dynamics and control on separate threads, though we have not yet found it advantageous to do that.

The FlightManager API contains a single virtual method, `update()`, which accepts the current time and the state of the vehicle (as computed by the dynamics), and returns the current motor values. The motor values are then passed to the dynamics object, which computes the new vehicle state. On the main thread, UE4's `Tick()` method queries the flight manager for the current vehicle pose (location, rotation) and displays the vehicle and its environment kinematically at the 60-120Hz frame rate of the game engine. In a similar manner, the threaded *VideoManager* classes can be used to process the images collected by a simulated gimbal-mounted camera on the vehicle, using OpenCV (Bradski, 2000). An abstract C++ *Target* class supports modeling interaction with other moving objects having their own dynamics; for example, in a predator/prey scenario.

This simplicity of our flight-control scheme makes it easy to connect MulticopterSim to existing flight-control software like Hackflight, or to the software-in-the-Loop (SITL) mechanism of ArduPilot (ArduPilot Dev Team, 2019b), as modules in the MulticopterSim codebase. With the Hackflight module, for example, we treat the control device (e.g., joystick, Xbox game controller) as "virtual receiver", which provides the R/C Receiver signal shown at the top of Figure 2. Further, the abstraction provided by Hackflight for sensing and open-loop control allows rapid prototyping of hybrid control systems, as we describe in the next section.

4 Toward neuromorphic flight control

As a demonstration of our approach, we used the Python-based Nengo neural simulator (Bekolay et al., 2014) to create a simple PID controller class for altitude hold. As shown in Figure 4, the controller consists of three populations of 200 spiking neurons: one population for computing the error between the target altitude and current altitude (P term); one for integrating the error (I term), and one for computing the error derivative D) term. (For this simple experiment we used only P.) The constants K_P, K_I, and K_d are implemented as arguments to the `transform` parameter of the `nengo.Connection` constructor; i.e., as connection weights between pools of neurons. We set the simulation time to 0.001 seconds, and used the default values for the remaining parameters in the Nengo class constructors. We made this Python class available to MulticopterSim by writing a C++ extension, allowing us to call the Python class (and hence the Nengo simulator) directly from C++.

For this trial experiment, the altitude-hold component was the only part of the FlightManager implementation that used Nengo; all other components used the Hackflight module, allowing us to fly the simulated quadcopter in the normal way (with game controller, joystick, or R/C transmitter with USB adapter), then triggering altitude-hold by pushing a button or flipping a switch on the controller.

To evaluate the performance of our neuromorphic PID controller we followed the same procedure as we use to evaluate other PID controllers in our simulator: raise the throttle to fly the vehicle to sufficient altitude (typically three or four meters), level off the throttle until the vehicle is relatively motionless, then flip the switch to enter altitude-hold mode. Using this procedure, we were able to keep the vehicle hovering, albeit with some vertical oscillation.

Although this PID controller has been hand-tuned by us to work with our simulator, and shows mediocre performance[3], it provides a simple proof of the feasibility of using an advanced neural simulator like Nengo in a real-time flight simulator, paving the way for more interesting experiments.

5 Conclusion and future work

As the closest robotic approximation to flying insects, birds, and mammals, miniature aerial vehicles (MAVs) offer a compelling new platform for research

[3]A better method for altitude hold is to use the error between the target and actual altitudes as a set-point for a secondary, velocity-based PID controller, as is done for example in ArduPilot.

in neuromorphic sensing, notably in the realm of vision (Mitrokhin, Sutor, Fermüller, & Aloimonos, 2019). Such research faces unique challenges.

In the physical realm, the current weight and form factor of event-based dynamic vision sensor (DVS) devices makes them impractical for deployment on micro-scale aerial vehicles.

In simulation, the 60-120 Hz frame rate of game engines like UE4 and Unity (Menard, 2011) exceeds that of most commercially-available CMOS cameras but is inadequate for emulating the multi-kilohertz data rates enabled by DVS (Gallego et al., 2019). Hence, one of our current research directions involves modeling the DVS datastream directly from the dynamics of the vehicle and target object.

Finally, to extend our Nengo-based PID controller in a more biologically realistic direction, we are experimenting with a Python version of our multirotor dynamics code, to exploit Nengo's support for reinforcement learning (Bekolay & Eliasmith, 2011). This paradigm would provide an accelerated way to develop neuromorphic flight controllers in an abstract mathematical simulation, to be validated by transferring them to MulitCopterSim, and eventually to an actual vehicle.

For both real and simulated flying robots, we see our minimalist, integrated approach to software and firmware design as a promising direction for robust aerial neurorobotics.

Table 2: Approximate sizes of two flight-simulation packages[5]

Package	Lines of code
AirSim	74,106
MulticopterSim	1,535

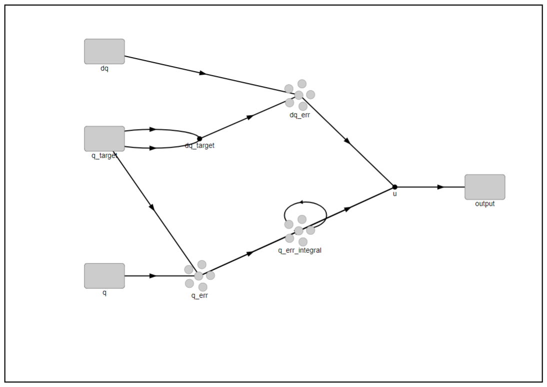
6 Downloads

The software described in this paper can be downloaded from the following repositories:

- `https://github.com/simondlevy/Hackflight`
- `https://github.com/simondlevy/MulticopterSim`
- `https://github.com/simondlevy/MulticopterSim/tree/NengoModule`

[5]The line count for MulticopterSim includes the module for Hackflight (see main text for details).

Figure 4: Nengo model for simple PID control



Acknowledgments

We thank Terry Stewart for help with the Nengo PID controller, and Shital Shah for the header-only rewrite of Hackflight. This research was supported by winter 2019 sabbatical-leave funding from Washington and Lee University.

References

ArduPilot Dev Team. (2019a). *History of ardupilot.* (http://ardupilot.org/planner2/docs/common-history-of-ardupilot.html, Accessed 15 June 2019)

ArduPilot Dev Team. (2019b). *Sitl simulator (software in the loop).* (http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html, Accessed 17 June 2019)

Banzi, M., & Shiloh, M. (2014). *Getting started with arduino.* MakerMedia.

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., et al. (2014). Nengo: A python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, *7*(48).

Bekolay, T., & Eliasmith, C. (2011). A general error-modulated stdp learning rule applied to reinforcement learning in the basal ganglia. *Cognitive and Systems Neuroscience.*

Bouabdallah, S., Murrieri, P., & Siegwart, R. (2004). Design and control of an indoor micro quadrotor. In *Proceedings of the 2004 IEEE international conference on robotics and automation, ICRA 2004, april 26 - may 1, 2004, new orleans, la, USA* (pp. 4393–4398).

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools.*

Cleanflight Team. (2019). (http://cleanflight.com, Accessed 15 June 2019)

E. Rohmer, M. F., S. P. N. Singh. (2013). V-rep: a versatile and scalable robot simulation framework. In *Proc. of the international conference on intelligent robots and systems (iros).*

Floreano, D., Zufferey, J.-C., Srinivasan, M. V., & Ellington, C. (2009). *Flying insects and robots* (1st ed.). Springer Publishing Company, Incorporated.

Gallego, G., Delbrück, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., et al. (2019). Event-based vision: A survey. *CoRR*, *abs/1904.08405.*

Gibson, J. J. (1979). *The ecological approach to visual perception.* Houghton Mifflin.

Menard, M. (2011). *Game development with unity* (1st ed.). Boston, MA, United States: Course Technology Press.

Mitrokhin, A., Sutor, P., Fermüller, C., & Aloimonos, Y. (2019). Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception. *Science Robotics*, *4*(30).

Sanders, A. (2016). *An introduction to unreal engine 4.* Natick, MA, USA: A. K. Peters, Ltd.

Shah, S., Dey, D., Lovett, C., & Kapoor, A. (2017). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics.*