Flying Blind with Reactive Control of Aerial Drones

Luca Mottola*+ and Kamin Whitehouse†
*Politecnico di Milano (Italy), +SICS Swedish ICT,
† University of Virginia, US

Aerial drones represent a new breed of mobile computing. Compared to mobile phones and connected cars that only opportunistically sense or communicate, they offer direct control *over their movements. Because of this, drones are enabling a range of sophisticated applications, such as photogrammetry and 3D reconstruction [12], as well as exploration of near-inaccessible areas [6].*

Aerial drones have also recently become fertile ground for challenging research problems. One example is the notion of *reactive control* for autonomous drones [5]. Reactive control is cast in the software design of current drone platforms, shown in Fig. 1. Two components are involved. Specialized software runs at a ground-control station (GCS) to let users configure mission parameters, such as the coordinates to cover through waypoint navigation. The GCS is typically a standard computer that communicates with the drone using a long-range radio.

Aboard the drone, the *autopilot* software implements the low-level control in charge of autonomously steering the drone. The control loop processes various sensor inputs, such as accelerations and GPS coordinates, to operate the electrical motors that set the 3D orientation of the drone, also termed as the drone's *attitude*. In doing so, the autopilot's goal is minimize the error between the actual and desired pitch, roll, and yaw, shown in Fig. 2. Because of size, cost, and energy concerns, autopilots run on resource-constrained embedded hardware.

Since we started working with aerial drones for mobile computing, we often found the performance of existing autopilot implementations to be somewhat

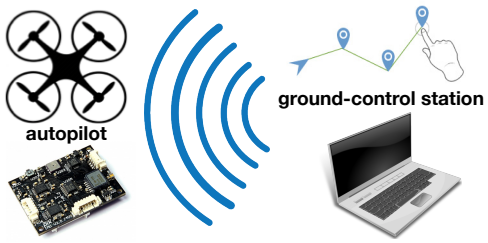


Figure 1: Software components in mainstream drone platforms.

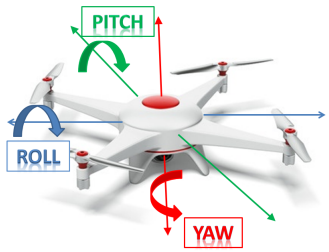


Figure 2: Attitude control with raw, pitch, and yaw.

disappointing. Then, we looked more closely at how autopilots work, at how they are implemented, and at the kind of hardware they typically run on, and eventually realized two essential aspects:

1. Most autopilot implementations employ Proportional-Integral-Derivative (PID) [2] designs: every T time units, sensors are probed, control decisions are computed, and commands are sent to the motors. However, the controllers are often tuned so that it is mostly the *Proportional* component to bear an influence. If the weights are balanced, the Derivative component can be kept to a minimum [4, 8]. A proper calibration of navigation sensors may also reduce the impact of the Integral component [4, 8].

2. Autopilot software typically relies on sensing hardware that closely resembles mobile phones [1]. Such sensors are especially designed to enable energy-efficient high-frequency sensing; for example, for tracking human activity [10]. Many of them can also be programmed to return a value only upon verifying certain conditions; for example, when a threshold is passed, to implement functionality such as fall detection [10].

The implications of these observations are profound. The first observation entails that the control loops unfold in ways where: *i)* small variations in the sensor inputs tend to correspond to small variations in the motor settings, and *ii)* as long as the sensor inputs do not change, the motor settings remain almost unaltered. Therefore, in principle, one may spare control executions that start from the same or similar sensor inputs as the previous iteration, simply maintaining the earlier motor settings. In a sense, the drone would fly without considering the navigation inputs from sensors, that is, it would be *flying blind*.

In practice, detecting the conditions when the drone can fly blind, as opposed to when it needs to promptly *react* to new conditions, requires algorithms, software implementations, and underlying hardware support that together incur smaller processing overhead and energy consumption than simply running the control loop. The second observation above is, in fact, a stepping stone in terms of hardware support, as modern sensors are extremely energy efficient. In the following, we describe the necessary algorithms and software support.

[1] goo.gl/SPOIR

Reactive control provides several advantages, for example: *i)* it enables more timely and adaptive control decisions, *ii)* it spares unnecessary processing, improving the utilization of the hardware, and *iii)* it lessens the need to overprovision control rates to handle extreme situations. For example, our results indicate that reactive control obtains up to 41% improvements in the accuracy of motion, which results in up to 22% extension of flight times [5]. Due to the limited lifetime of current drone technology, the latter are particularly valuable.

An Example Autopilot

Ardupilot[2] is a paradigmatic example of a mature open-source project that provides reliable autopilot functionality [3]. It is at the basis of many commercial products [4] and boasts a large on-line community.

The execution of Ardupilot's control loop is split in two parts. The so-called *fast loop* only includes the implementation of PID controllers for critical motion control. The time left from the execution of *fast loop* is given to an application-level *scheduler* that distributes it among non-critical tasks, such as logging. The scheduler operates in a *best-effort* manner. Many autopilots share similar designs [5].

In Ardupilot, the execution rate is statically set based on a few "rules of thumbs" [15]. The *fixed* 400 Hz setting on the hardware we describe next, however, is not necessarily the maximum the hardware supports, as it is thought to leave enough room—on average—to the scheduler. In short bursts, *fast loop* may run much faster than 400 Hz, as long as some resources are eventually allocated to the scheduler; for example, at times when control does not need to run that frequently. By recognizing the situations when control does need to run—or not—we enable precisely this kind of dynamic adaptation.

Ardupilot supports various embedded hardware. A primary example is the Pixhawk board, which features a Cortex M4 core at 168 MHz and a full sensor array for navigation. These sensors have similar capabilities as those on modern mobile phones. They support energy-efficient high-frequency sampling and often provide interrupt-driven modes to generate a value upon verifying certain conditions. The ST LSM303D on the Pixhawk, for example, can be programmed to generate an SPI interrupt based on three thresholds. While useful, for example, for functionality such as fall detection [10], these features are rarely exploited in autopilots.

[2] At the start of the project, Ardupilot ran on Arduino hardware. However, developers eventually moved to more capable hardware while retaining the name.

[3] goo.gl/x2CHyM

[4] goo.gl/sBoH6

[5] goo.gl/uCGmr4,goo.gl/D89lkb.

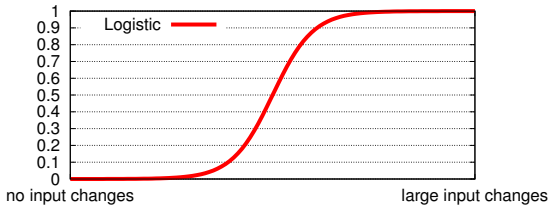


Figure 3: Example logistic function.

Flying Blind

Reactive control requires to address three issues. First is how to recognize whether the readings of navigation sensors require new control settings to be computed, and hence the control loop does need to run. Second is how to handle the possibly contradicting indications for running the control loop coming from different navigation sensors, at different rates, and asynchronously with respect to each other. In addition, the possibility that the control loop may not be running for too long and eventually impact the stability of the drone. Third is how to implement the resulting reactive processing on resource-constrained embedded hardware. We provide next a glimpse of how we address these issues; detailed descriptions are also available [5].

When to run control? It may seem intuitive that the more "significant" is a change in a sensor reading, the more likely is the necessity to run the control loop. Such a condition would indicate that something just happened in the environment that requires the drone to react. However, what is a "significant" change in the sensor readings depends on several factors, including the accuracy of sensor hardware, the physical characteristics of the drone, the actual control logic, and the granularity of the control outputs.

Our solution abstracts from these aspects: despite the control logic is deterministic, we consider a change in the control settings as a random phenomena. The input to this phenomena is the difference between consecutive samples of the same navigation sensor; the output is a binary value indicating whether the control settings need to change. If so, we need to run the control loop to compute the new settings. An accurate estimator of such phenomena would allow us to take an informed decision on whether to run the control loop.

Among statistical estimators with a *binary* dependent variable, *logistic regression* [9], shown in Fig. 3, closely matches the intuition above. For small changes in the sensor inputs, the probability of changes in the control settings is small. When changes in sensor inputs are large, a change in the control settings becomes (almost) certain. It also turns out it is possible estimate the parameters shaping the curve of Fig. 3 efficiently, because logistic regression allows one to employ traditional estimators, such as ordinary least squares.

When a drone starts, we run the control loop at fixed rate for a predefined limited time, tracking whether the control settings change. This gives us an initial data set to employ least square estimators to compute the parameters

of logistic regression[6]. Afterwards, for every change in the navigation sensors, logistic regression indicates how likely is a change in the sensor inputs to require new control settings. Comparing this probability against an adaptable threshold dictates whether to run the control loop.

The least square estimation may possibly repeat later throughout the execution as false positives (negatives) are identified, as part of the best-effort *scheduler* of Ardupilot. Moreover, to cater for situations where false negatives happen in a row, we run the control loop anyways at very low frequency, typically in the range of a few Hz. If such executions compute new control settings, the drone most likely applies some significant correction to the flight operation that causes reactive control to be triggered immediately after.

How to handle multiple sensors? PID controllers used in autopilots are conceived under the assumption that sensor are sampled almost simultaneously and at a fixed rate. In reality, the time of sampling and therefore of possibly recognizing the need to execute the control loop, is not necessarily aligned across sensors. Drastic changes in the sensor inputs may also be correlated. For example, when the accelerometers record a sudden increase because of wind gusts, a gyroscope also likely records significant changes. A traditional implementation would process these inputs together.

We take a conservative approach to address these issues. Based on the sampling frequency of every sensor in the system, we compute the system's *hyperperiod* as the smallest interval of time after which the sampling of all sensors repeats. Upon recognizing first the conditions requiring the execution of the control loop, we wait until the current hyperperiod completes. This allows us to "accumulate" all inputs on different sensors, giving the most up-to-date inputs to the control logic at once.

How to implement it efficiently? The control logic is implemented as multiple processing steps arranged in a complex multi-branch pipeline. Depending on what sensor indicates the need to execute the control loop, different slices of the code may need to run while other parts may not. Moreover, each such processing step may—in addition to producing an output *immediately* useful—update global state used *at a different iteration* elsewhere in the control pipeline.

In this setting, reactive control makes the processing event-driven also because of asynchronous updates to global state. Employing standard programming techniques in these circumstances quickly turns implementations into a "callback hell" [7]. This fragments the program's control flow across numerous syntactically-independent fragments of code, hampering compile-time optimizations. This causes an overhead that limits the benefits of reactive control [5].

We tackle this issue using a technique called *reactive programming* [3], rarely employed in embedded computing because of resource constraints. We create a reactive programming implementation tailored to the hardware we target, with additional custom semantics. Using reactive programming requires to refactor the implementations of autopilots. In our experience, using proper code

[6]Note that this design considers the initial drone execution as representative of the rest of the flight. Should this not be the case, a fail-over mechanism kicks in that recomputes the logistic regression parameters from scratch.

(a) Quadcopter. (b) Hexacopter.

Figure 4: Custom aerial drones for performance evaluation.

inspection tools [7], the required effort is quite limited. Re-factoring Ardupilot only required three days of work. Other autopilots required less time [5].

Final considerations. On the surface, reactive control may resemble the notion of event-based control [1]. In the latter, however, the control logic is expressly redesigned for settings different than ours; for example, in distributed control to cope with limited bandwidth. Our work aims at re-using existing control logic, whose properties are well understood. Different than event-based control, however, reactive control is mainly applicable only to PID-like controllers where the Proportional component dominates.

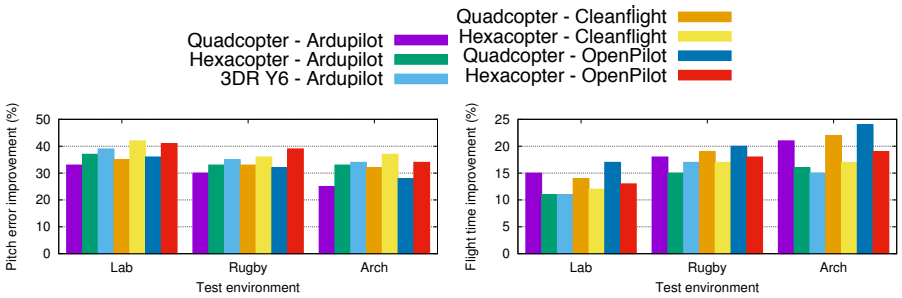
In the field of aerial drones, demonstrations exist showing motion control in tasks such as throwing and catching balls [13] or flying in formation [14]. In these applications, the low-level control does not operate aboard the drone. At 100 Hz or more, a powerful computer receives accurate localization data [8], runs sophisticated control algorithms based on mechanical models expressed through differential equations, and sends actuator commands back to the drones. Differently, we aim at improving the performance of mainstream low-level control running on embedded hardware.

Performance

We measured the performance of reactive control against the original implementation of Ardupilot as well as that of OpenPilot and Cleanflight. In the following, we describe an excerpt of the results we collect [5].

We use the two custom drones of Fig. 4 plus a 3D Robotics Y6 drone. The latter is peculiar as it is equipped with only three arms with two co-axial motor-propellers assemblies at each end, requiring a drastically different control logic. We test three environments: *i)* a 20x20 m indoor lab, termed LAB; *ii)* a rugby field termed RUGBY, using GPS; and *iii)* an archaeological site in Aquileia (Italy) termed ARCH [11]. The sites exhibit increasing environment influence, from the mere air conditioning in LAB to average wind speeds of 8+ knots in

[7] goo.gl/0VZGAc
[8] goo.gl/Vh5Q4c



(a) Pitch error improvement.

(b) Flight time improvement.

Figure 5: Performance improvements with reactive control.

ARCH. The variety of software, hardware, and test environments demonstrates the general applicability of reactive control.

Based on 260+ hours of tests, Fig. 5(a) shows the average improvements in pitch error; these are significant, ranging from a 41% reduction with Cleanflight in LAB to a 27% reduction with Ardupilot in the ARCH. We obtain similar results, sometimes better, for yaw and roll [5]. Comparing this performance with earlier experiments [5], we confirm that it is the opportunity to spare iterations of the control loop that enables more accurate control decisions. Not running the control loop unnecessarily frees resources, increasing their availability whenever there is actually the need to use them. In these circumstances, reactive control dynamically increases the rate of control, possibly beyond the pre-set rate.

Still in Fig. 5(a), the improvements of reactive control apply to the Y6 as well; in fact, these are highest in a given environment. This cannot be attributed to its structural robustness; the Y6 is definitely the least "sturdy" of the three. We conjecture that the different control logic of the Y6 offers additional opportunities to reactive control. A similar reasoning apply to Cleanflight, shown in Fig. 5(a). Being the youngest of the autopilot we test, it is fair to expect the control logic to be the least refined. Reactive control is still able to drastically improve the pitch error, by a 32% (37%) factor with the quadcopter (hexacopter) in ARCH.

The improvements in attitude error translate into more accurate motion control and fewer attitude corrections. As a result, battery utilization improves. Fig. 5(b) shows the results we obtain in this respect. Reactive control reaches up to a 24% improvement. This means flying more than 27 min instead of 22 min with OpenPilot in ARCH. This figure is crucial for aerial drones; the improvements reactive control enables are thus extremely valuable. Most importantly, these improvements are higher in the more demanding settings. Fig. 5(b) shows that the better resource utilization of reactive control becomes more important as the environment is harsher. Similarly, the quadcopter shows higher improvements than the hexacopter. The mechanical design of the latter already makes it physically resilient. Differently, the quadcopter offers more ample margin to cope with the environment influence in software.

Conclusion

Reactive control replaces traditional autopilot control by governing the execution of the control logic based on changes in the navigation sensors. This allows the system to dynamically adapt the control rate to varying environment dynamics. To that end, we conceived a probabilistic approach to trigger the execution of the control logic, a way to carefully regulate the control executions over time, and an efficient implementation on resource-constrained hardware. The benefits provided by reactive control include higher accuracy in motion control and longer operational times.

Acknowledgments. This work has benefited from the talents and hard work of many students, including Endri Bregu, Daniel Cantoni, and Nicola Casamassima. The work was partly supported by the Projects "Zero-energy Buildings in Smart Urban Districts" (EEB), "ICT Solutions to Support Logistics and Transport Processes" (ITS), and "Smart Living Technologies" (SHELL) of the Italian Ministry for University and Research.

References

[1] K. J. Åström. Event based control. In *Analysis and Design of Nonlinear Control Systems*. Springer Verlag, 2007.

[2] K. J. Åström and T. Hägglund. *Advanced PID control*. ISA - The Instrumentation, Systems, and Automation Society, 2006.

[3] E. Bainomugisha et al. A survey on reactive programming. *ACM Comput. Surv.*, 45(4), 2013.

[4] S. Bouabdallah, A. Noth, and R. Siegwart. PID vs LQ control techniques applied to an indoor micro quadrotor. In *Proceedings of IROS*, 2004.

[5] E. Bregu, D. Cantoni, N. Casamassima, L. Mottola, and K. Whitehouse. Reactive control of autonomous drones. In *Proceedings of ACM MOBISYS*, 2016.

[6] W. Burgard et al. Collaborative multi-robot exploration. In *Proceedings of ICRA*, 2000.

[7] J. Edwards. Coherent reaction. In *Proceedings of the ACM Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, 2009.

[8] R. M. Faragher et al. Captain Buzz: An all-smartphone autonomous delta-wing drone. In *Workshop on Micro Aerial Vehicle Networks, Systems, and Applications (colocated with ACM MOBISYS)*, 2015.

[9] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

[10] E. Miluzzo et al. Sensing meets mobile social networks: The design, implementation and evaluation of the CenceMe application. In *Proceedings of ACM SENSYS*, 2008.

[11] L. Mottola et al. Team-level programming of drone sensor networks. In *Proceedings of ACM SENSYS*, 2014.

[12] F. Nex and F. Remondino. UAV for 3D mapping applications: A review. *Applied Geomatics*, 2003.

[13] R. Ritz et al. Cooperative quadrocopter ball throwing and catching. In *Proceedings of IROS*, 2012.

[14] M. Turpin, N. Michael, and V. Kumar. Decentralized formation control with variable shapes for aerial robots. In *Proc. of ICRA*, 2012.

[15] M. Zhuang and D. Atherton. Automatic tuning of optimum PID controllers. *IEEE Proceedings on Control Theory and Applications*, 140(3), 1993.