Mobile Systems Research with Drones

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

Robot vehicle platforms, often called "drones", offer exciting new opportunities for mobile computing. While many systems respond to device mobility (such as smartphones), drones allow computer systems to actively control device location, allowing them to interact with the physical world in new ways and with new-found scale, efficiency, or precision.

The startup cost to experiment with and build real drone applications has dropped dramatically in recent years, also thanks to technological developments driven by the smartphone industry and the rise of the "makers" and DIY movements. As with any emerging technology, however, a fragmented software and hardware ecosystem can leave newcomers wondering where to start.

This paper provides an overview specifically for researchers who want to explore the mobile computing challenges made available by drones, such as reliability, energy management, mobile communication, and programmability. Building on several years of research using drones for mobile computing—from tiny quadcopters weighing 20 grams to octocopters like the one in Fig. 1 and planes powered by combustion engines—we outline key points of entry into the drone ecosystem, including several advantages and potential pitfalls.

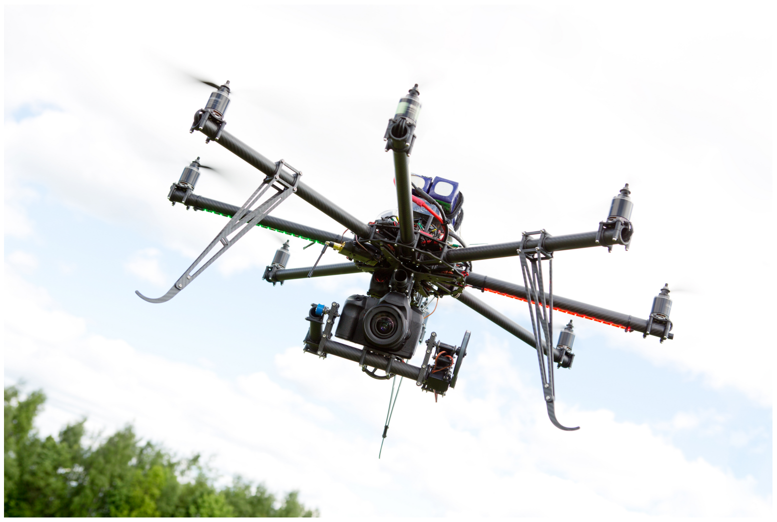

Figure 1: Aerial drone: octocopter.

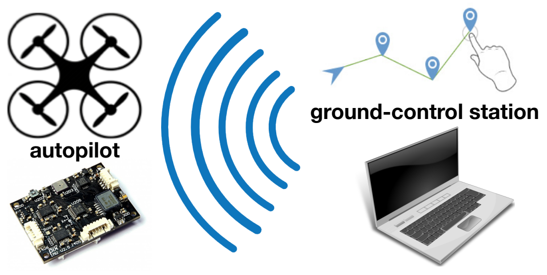

Figure 2: Software components in mainstream drone platforms. *The ground control station (GCS) lets users configure high-level mission parameters, the autopilot software implements the low-level motion control aboard the drone.*

Figure 3: Remote controller for manual flight.

Drones...what?

Drone applications are often designed with a two part architecture in which a single, centralized computer systems tasks each of the drones, which execute their tasks autonomously, as illustrated in Fig. 2. Specialized software runs at a *ground-control station (GCS)* to let users configure mission parameters, such as the coordinates to cover through waypoint navigation and the action to take at each waypoint. The GCS is typically a standard computer that communicates with the drone using Wi-Fi, Bluetooth, or a long-range low-bandwidth radio. Unlike Wi-Fi and Bluetooth, the latter most often work in the 433 Mhz or 915 Mhz bands, and may cover up to a few kilometers.

Aboard the drone, the *autopilot* software implements the low-level control to autonomously navigate to the next waypoint. The control loop processes various sensor inputs, such as accelerations and GPS coordinates, to operate the motors that set the 3D orientation of the drone, also called the drones *attitude*. Because of size, cost, and energy concerns, autopilots run on resource-constrained embedded hardware.

Whether to focus on the GCS or on the autopilot depends on the scope of the project. At the GCS it is possible, for example, to implement high-level functionality; for example, to coordinate multiple drones for a given application [9]. In contrast, the autopilot functionality includes low-level control loops to steer the drone based on high-level inputs provided by the GCS. The autopilot functionality often does not include any application-specific logic, although there are many opportunities for cross-layer optimization between the GCS and autopilot. The latter may also be used without a GCS, using a remote controller such as the one in Fig. 3 for manual control.

Software platforms for the GCS and autopilot have become widely accessible. Sophisticated solutions exist both in the open-source and open-hardware domains, as well as in the commercial landscape. The former naturally offer more flexibility because anything can be customized or replaced all together, including GCS software, autopilots, and the underlying hardware. However, the learning curve for open platforms can be steep because they need to be configured, assembled, and calibrated from scratch. This often entails choosing each and every component from a vast and sometimes confusing catalogue of frames, motors, propellers, autopilot software and hardware, and the like. It may thus take some time (and several crashes) before being able to fly reliably. On the other hand, commercial drone platforms often come "ready-to-fly", but customization is often limited to only the GCS software.

Available drone platforms already enable a wide range of real-world applications that would be unfeasible or impractical with any other technology, such as precision agriculture, aerial photogrammetry, 3D reconstruction, disaster management, and surveillance applications [2,5,7,10], as illustrated in Fig. 4. Most of these applications require outdoor operation where positioning can be obtained from GPS. However, these complex applications also require significant manual intervention and achieving a fully autonomous behavior is still a challenge.

Figure 4: A hexacopter performing 3D reconstruction in Egypt.

Many demonstrations have shown drones performing sophisticated autonomous tasks such as throwing and catching balls [14] or cooperatively carrying large payloads [8]. However, the goal of these demonstrations is to explore new approaches for mechanical motion control, and they still require extremely accurate positioning systems [15], significant processing power at the GCS, and highly-instrumented lab settings or highly customized platforms whose mechanical design is strictly coupled with hardware/software components. The challenges involved in building a fully autonomous system—possibly including multiple cooperating drones—to complete complex applications in natural, outdoor, and uninstrumented environments are still relatively unexplored.

Application control

Even with a commercial drone platform that does not offer great customization opportunities, a wide range of research possibility is within reach. High-level APIs and supporting communication protocols are available to steer drones remotely, from a PC or even a tablet or a smartphone. We believe this is, in fact, one of the best ways to start working with drones. Besides sparing the effort for configuration, assembly, and calibration, commercial drones offer simple, yet useful emergency features; for example, pre-programmed maneuvers in case a drone ended up entangled in hard-to-reach places, such as trees.

DroneKit by 3DRobotics (`dronekit.io`) is an example framework enabling high-level control. It offers a Python API including high-level commands to steer the drone independent of the vehicle platform, from quadcopter to fixed-wing planes. Applications developed this way may add greater intelligence to the drone's behavior and perform tasks that are computationally- or time-sensitive, such as computer vision and path planning. In addition, these frameworks enable the integration of drone-specific processing with larger cloud-based applications through REST-based interactions or messaging protocols such as MQTT.

Underneath DroneKit and similar frameworks is a standard way to exchange flight commands and status information with the drone, called MAVLink [13]. This is an open protocol for communicating with small unmanned vehicles, designed as a header-only message marshalling library. It is also widely adopted: many open-source and open-hardware platforms, as well as commercial systems can speak MAVLink. Those interested in realizing high-level applications with drones, yet unhappy with the aforementioned programming frameworks, may decide to implement their own framework on top of MAVLink by designing the programming abstractions that best suit their target application domain. Alternatively, they may simply let their application talk MAVLink directly, thus gaining in efficiency but sacrificing generality.

High-level application processing, no matter the software framework, does not necessarily need to run on the ground. The availability of extremely inexpensive single-board computers has led to the emergence of the "companion computer" as a viable alternative to a GCS. DroneKit and similar frameworks are capable to run on any RaspberryPI or BeagleBone-equivalent, which is sufficiently small and lightweight to be physically loaded on the drone, making the system completely independent of external infrastructure. The energy required to operate the companion computer is, in most cases, much smaller than the energy spent to power the motors, so it can be hooked to the main power source aboard the drone without being overly detrimental to the drone's lifetime. On the other hand, the application processing–especially if spanning multiple drones—and supporting programming frameworks need to be tailored to the resources available on such single-board devices, where memory and processing power cannot be taken for granted.

Low-level control

Together with the mechanical design, the autopilot software determines the effectiveness of physical motion. For example, when using drones in imagery applications, the low-level control directly influences the quality of the shots [10]. Further, the low-level control is partly responsible for how the energy available from batteries is consumed, because the drones lifetime is often a result of streamlined mechanical operation [3].

Most existing autopilot implementations employ Proportional-Integral-Derivative (PID) [1] designs for low-level control. These controllers run in a time-triggered fashion: every T time units, navigation sensors such as GPS, accelerometers, and gyroscopes are probed, control decisions are computed by dedicated hardware, and commands are sent to the actuators to operate the motors. Such an approach enjoys the advantage of highly deterministic operation, which simplifies implementation.

Although the autopilot implementations aboard commercial drones are typically closed-source and cannot be customized, a number of mature open-source autopilot projects are readily available. Among them, Ardupilot (`ardupilot.org`) is a prominent example. The project is at the basis of a large community and often comes pre-installed onto many drone platforms, including those of 3DRobotics and many others. Other examples are Cleanflight (`cleanflight.com`) and OpenPilot (`openpilot.org`). Moreover, while ArduPilot is explicitly designed for autonomous flight, autopilot implementations exist that are especially optimized for manual control, including "first-person view" (FPV) operation and aerial acrobatics. In this case, the autopilot also offers support for streaming high-quality video to a headset, typically through an additional radio running custom protocols in the 5.8 GHz band.

Most open-source autopilot implementations typically support a few underlying hardware platforms; in turn, the latter often come in the form of open-hardware designs and are particularly optimized for a given autopilot implementation. For example, in the case of Ardupilot, the ideal hardware platform is the Pixhawk board [12], shown in Fig. 5, which features a Cortex M4 core at 168 MHz and a full sensor array for navigation, including a 16-bit gyroscope, a 14-bit accelerometer/magnetometer, a 16-bit 3-axis accelerometer/gyroscope, and a 24-bit barometer. Most often, at least a sonar and a GPS are added to

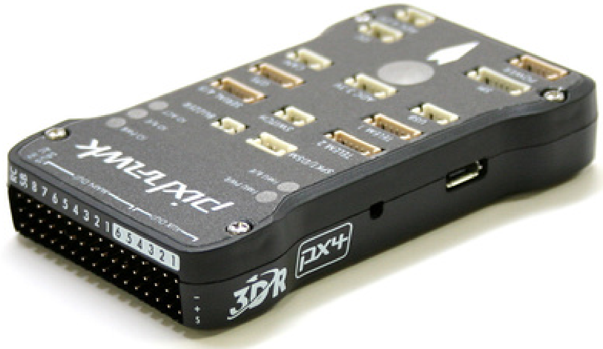
Figure 5: Pixhawk board for drone autopilots.

the built-in sensor array to provide positioning and altitude information.

Interestingly, much of this hardware—and especially the sensor equipment—appears to be inherited from smartphones, mostly with a three- to four-year lag. For example, the Invensense MPU 6000 3-axis accelerometer/gyroscope on the Pixhawk was seen on a range of HTC smartphones; the ST Micro LSM303D 14-bit integrated accelerometer-magnetometer on the Pixhawk was used in Samsung's low-range smartphone offering. Here is where drones have benefited the most from push of the smartphone industry; *some argue that without modern smartphones, we would not have drones in their current form [6].*

Working at a high level may appear to offer more readily-available opportunities for researching new problems, compared to dealing with low-level concerns rooted in decade-old control literature and embedded hardware. Our experience tells a different story. Several seemingly-solved problems must be revisited in the new context set by drones and the applications they enable, including control [4]. Moreover, it is at this level that one of the most severe issues still plaguing the operation of drones, that is, their dependable behavior, needs to be tackled [11], as we discuss next.

Lessons learned

Over the past five years, we learned several lessons that may be useful for mobile systems researchers interested in new projects that involve drones:

Break (your) stuff. We started with drones using the AR.Drone 2.0, a commercial drone platform provided by Parrot. The AR.Drone was, and still is, among the cheapest drone platforms that still allow for credible research and experimentation. The AR.Drone 2.0 can speak MAVLink. A C++ SDK is available from Parrot to implement application-level remote control, while several open-source alternatives in other languages also exist. Most importantly, spare parts are readily available. Breaking bits and pieces was thus not much of an issue, which lets one experiment with things one would not try if there were hundreds or even thousands of euros at stake. Being "brave" is fundamental to gain experience with these devices.

Do not break somebody else's stuff (or people). The AR.Drone was so easy to fly that it appeared to be safe to do so even indoor or in confined spaces. We quickly realized, however, how dangerous this could become. The AR.Drone propellers spin hundreds of times per minute, the device weights about .5 kg, and it may accelerate up to 10 m/s in a straight line. Hitting an object, or even a person, in these conditions may cause serious accidents. Independent of the regulations currently in force in one's country, and even though a practical "drone testbed" is yet to be seen, researchers must be sure that a protected area of reasonable size is available for experimentation.

Fly manual. While our initial research focused mainly on autonomous flight at the level of application control, flying the AR.Drone manually turned out to be extremely valuable. It made us understand more precisely how the drone reacts to external commands, which facilitated debugging the behavior during autonomous flight; how the environment may affect its operation, for example, in the case of wind; and what signs indicate that physical breakage is imminent, which turns out to be fundamental for a dependable operation in the wild. Some of our research in this area [4] was only possible because of the experience gained in flying drones manually.

The more sensors, the better. When we transitioned from the AR.Drone to more custom-designed platforms, we needed to decide the best combination of components for application needs. While it may be tempting to reduce cost by reducing the set of navigation sensors, this is often a big mistake. Autopilots are complex pieces of software, and their use of sensor inputs is often not obvious. For example, some autopilots employ two GPS receivers not as a failover measure, but as parallel receivers used during normal operation, which drastically increases the precision of navigation. This lead to unexplained frustration with our early custom vehicle platforms, which were equipped with only a single GPS receiver. When creating a customized platform, we recommend over-provisioning first and performing the cost engineering in later stages.

Take good care of them. Drones are not like smartphones, where the worst thing that may happen during routine use is the battery being discharged. Think of them as a bicycle: they need continuous maintenance. The integrity of the frame, the proper attachment and rotation of motors and propellers, as well as the conditions of the battery must be checked *before every flight*. We learned this the hard way in a deployment in an archaeological site in Italy [9]. Archaeological sites are particularly dusty environments and, as dust enters a drones gears and shafts, their efficiency quickly drops until the drone fails to operate. We eventually identified a suitable maintenance schedule but, in the meantime, we broke four motors, eleven propellers, and uncountable gears. Interestingly, at least in our experience, mechanical failures on drones are seldom totally unanticipated. If it makes an unusual noise when taking off, land it immediately and thoroughly re-check everything.

Research directions

Drones have suddenly created a cost-effective mobile computing platform whose physical behavior in time and space is under complete control. What research opportunities does that offer? There are probably many more than what we can hint here, spanning essentially every aspect of a computing system, from hardware to application layers. Drones also offer new connections and opportunities with fields other than mobile computing, such as control and mechatronics, that can be explored.

For example, achieving fully autonomous behaviors is, in large part, still an open challenge. Artificial intelligence and robotics have been making progress in these areas for decades, mainly focusing on general-purpose solutions whose system implementations are, however, often hardware-specific. In contrast, the mobile computing community can offer a system foundation to build upon. It should provide the necessary functionality and performance in a way to simplify the design and implementation of intelligent algorithms, while enabling portability of implementations across devices.

Tackling the challenge of autonomous behavior becomes even more difficult when distributed coordination among multiple drones is necessary or simply beneficial to the application. A range of questions then arise: with each drone equipped with a limited energy budget, how do we manage the available system-wide energy in a way to maximize the application objectives? How do we manage network connectivity to ensure that drones have the necessary communication support to enable the coordination? What kind of distributed data consistency models are most suited to a highly mobile scenario, and what can we sacrifice in overall correctness to gain in net performance? In case system-wide network connectivity cannot be guaranteed, how do we assign tasks to individual drones in a way that is robust to communication failures?

The lack of proper abstraction layers in current drone platforms, which results in laborious and one-off development processes, represents a further orthogonal challenge. The idea of installing “apps” on a drone with the same ease as on smartphones, while fascinating, is still a long way ahead. New and wide-reaching infrastructure would need to be built towards that end, both on the drones themselves and in terms of development and deployment support.

Further, problems that are relatively simple to solve on mainstream computing platforms may become extremely difficult on drones, because the dependability requirements are brought to an extreme. Accidents with drones are often reported whose causes may be attributed to malfunctioning software or hardware, like the case of the camera drone almost hitting a skier during the world championship, shown in Fig. 6. Size, cost, and energy concerns, however, require drones to employ resource-constrained embedded hardware, which only allows developer to employ low-level languages and forces them to aggressively optimize implementations. Developing drone code is thus extremely error-prone. Although we are not necessarily advocating the extremely rigid methodologies used in avionics, a more principled approach to ensure dependability guarantees is certainly needed.

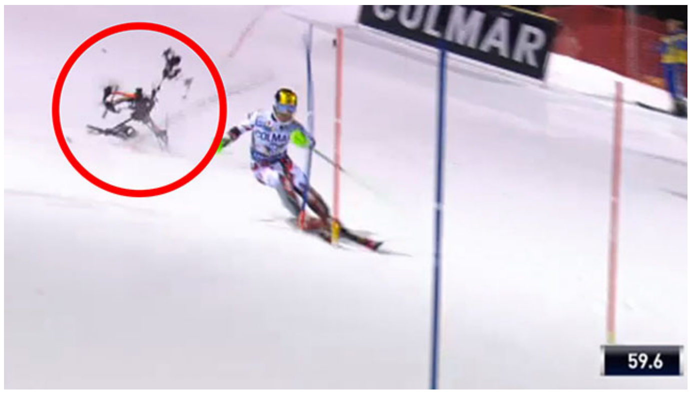

Figure 6: An accident where a camera drone crashed close to a skier. *Investigations later revealed that a malfunctioning battery sensor probably caused the accident. The human operator realized the potential failure too late.*

As much as we are starting to see “smartphone testbeds” appearing, the issues above would be ameliorated with better simulation support and easily-accessible “drone testbeds” that enable pre-deployment testing and verification. As of now, however, simulation support appears either limited in the amount of realism or focused on specific functionality, such as obstacle avoidance. The former limitation may be tackled with trace-driven simulations; however, such an approach becomes difficult or even impossible for systems that involve control, because control actions would introduce discrepancies between the trace and the simulated environment. The lack of generality also applies to the few examples of real-world testbeds, which are often platform-specific and tend to be complex to employ. Ideally, simulators and testbeds should expose the same software stack as real systems, enabling smoother transitions between the testing environments and real deployments.

Arguably, one of the main limitations of current drone platforms is still their lifetime. While we cannot expect battery technology to drastically change this in the near future, we believe interesting avenues for research are found in both optimizing the hardware designs for low-power operation of drones and in novel power provisioning systems. Early experimentation demonstrate, for example, how wireless energy transfer may be used to power tiny quadcopters remotely, indefinitely prolonging their lifetime.

Finally, a note is in order about regulations. While this may seem a non-technical problem, we argue the reason why many countries still do not have a stable regulation regarding the use of drones is because many technical aspects of existing platforms are misunderstood. *As researchers, it is also our job to help legislators gain a sufficient, un-biased vision on the technology.* At the same time, it is likely that we will find interesting problems to work on also in this area. At the time when we will develop an actual drone traffic management system, for example, one will require algorithms to automatically govern the traffic while ensuring the safety of operation, along with exchangeable data formats for real-time updates of current conditions across national borders.

Go have fun!

We only scratched the surface of what drones are, what can be done with them, what interesting research problems they offer, and how to tackle them. We believe there is much more to it. The best way to find it out is to just set yourself and your colleagues free from any specific research agenda and play with the technology. The possibilities are so many that you will likely find a problem that appeals to you, and the reward of being able to demonstrate your solution with a real system will be well worth the effort of tackling it.

Acknowledgments. Our work with drones has benefited from the talents and hard work of many students, including Mikhail Afanasov, Endri Bregu, Daniel Cantoni, Nicola Casamassima, Mattia Moretta, and Andrea Patelli. 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 and T. Hägglund. *Advanced PID control.* ISA - The Instrumentation, Systems, and Automation Society, 2006.

[2] BBC News. Disaster drones: How robot teams can help in a crisis. goo.gl/6efliV.

[3] G. A. Bekey. *Autonomous robots: From biological inspiration to implementation and control.* The MIT Press, 2005.

[4] E. Bregu et al. Reactive control of autonomous drones. In *Proceedings of ACM MOBISYS*, 2016.

[5] A. Bürkle. Collaborating miniature drones for surveillance and reconnaissance. In *Europe Security+ Defence*, 2009.

[6] C. Anderson. How I accidentally kickstarted the domestic drone boom. goo.gl/SPOIR.

[7] IEEE Spectrum Online. SenseFly and DroneAdventures toss UAVs off the summit of Matterhorn. goo.gl/N6ekAA.

[8] N. Michael et al. Cooperative manipulation and transportation with aerial robots. *Autonomous Robots*, 30(1), 2011.

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

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

[11] A. Patelli et al. Model-based real-time testing of drone autopilots. In *Proceedings of DRONET*, 2016.

[12] PixHawk.org. PX4 autopilot. goo.gl/wU4fmk.

[13] QGroundControl. MAVLink: Micro Air Vehicle Communication Protocol. goo.gl/fMPw0D.

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

[15] Vicom. Motion capture systems. goo.gl/Vh5Q4c.