Understanding the Software and Hardware Stacks of a General-Purpose Cognitive Drone

Sam Jijina, Adriana Amyette, Nima Shoghi, Ramyad Hadidi, Hyesoon Kim
Georgia Institute of Technology, Atlanta, GA
{sam.jijina, adriana.amyette, nimash, rhadidi, hyesoon.kim}@gatech.edu

***Abstract*—Fully autonomous drones have a plethora of applications in the real world, from agriculture and communication to public services. With increasing attention, a new market segment has opened up for highly efficient drones. However, the deployment of efficient drones requires an in-depth analysis of several components spanning from hardware sensors to software stack. Specifically, to achieve high reliability, safety, and performance, the top concerns in the professional drone industry are characterizing underlying architecture and flight stack. In this paper, we characterize a widely-used open source flight stack, ArduCopter, to understand the performance requirements as a research community. Additionally, we study how area-specific applications affect flight stack. Our characterizations and benchmarks indicate that the drone flying range can be dramatically increased by optimizing the underlying flight controller software.**

I. Introduction

Drones are rapidly expanding in several industries and serve a vast variety of markets [1]. Quadcopters, the most common type of drone, have an innate advantage by using pairs of clockwise and counter-clockwise rotating propellers which enable a robust drone for several applications. Currently, drone software and hardware stacks are highly specialized as an end-to-end system. For example, the SkyDio [2] drone is specialized for video and photography. Although such end-to-end specialization enables proprietary optimizations, it does not provide flexibility on the user end. In other words, current commercial drones rely so heavily on their proprietary software and systems that it limits multi-functionality. Despite commercial efforts, drones are still under experimentation and limited use. Hence, architecting the end-to-end system still remains an open research question with several opportunities.

To explore drone architecture and its software stack, we built a fully open source drone with commonly available hardware and sensors, shown in Figure 1a. Our drone uses the Linux operating system which allows us to execute the autopilot software as a service while leaving the system available to other programs such as cognitive services (e.g., video recording, human recognition, path planning). Further, using Linux as the base OS, we are able to have full control of the firmware, allowing for performance optimizations. Having full access to the drone's underlying hardware and software allows for easier expansion into connected drones and drone-to-drone communication. In detail, our drone is an open source quadcopter controlled by a Raspberry Pi running an open source flight control stack called ArduCopter [3]. The controller we use is the Navio2 [4] HAT developed for the Raspberry Pi [5]. The OS that we use is a modified Linux Kernel built with the RT-Preempt patch. For the characterization,

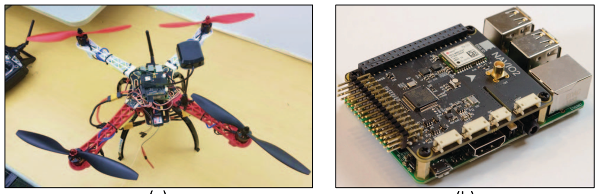

Fig. 1. (a) Our open source Raspberry-Pi-based drone. (b) The flight controller [4] mounted on the Raspberry Pi.

we measure the overall performance of the autopilot software and the drone while running tests along with other heavy cognitive-based workloads such as OpenCV-based applications or Simultaneous Localization and Mapping (SLAM) [6].

II. Drone Architecture

A. Hardware

The hardware consists of sensors and four motors. The physics of the rotational direction of opposite-pairs of propellers allows the drone to make varied movements in 3D space by changing PWM outputs to the motors. This decreases processing burden on the Raspberry Pi and allows us to use the Raspberry Pi for other activities such as flight control algorithms or high-level machine learning algorithms.

Flight Controller: The flight controller used for this prototype is the Navio2 [4] open source controller, shown in Figure 1b [5]. This controller is a HAT for the Raspberry Pi and interfaces with the Raspberry Pi using the GPIO pins. The Navio2 is pre-configured with GLONASS and GPS abilities and comes equipped with multiple IMUs to support guided navigation. The flight controller is the interface between the Linux Kernel and the hardware devices of the drone. The Raspberry Pi sends signals to the flight controller to be decoded. The controller then translates the instructions into PWM signals and then outputs signals to the four motors.

B. Software

The drone uses the open source ArduCopter [3] to carry out primary flight functions such as altitude and position control. We also integrated the open source DroneKit APIs [7] into the drone code base so that user programs written in C++ and Python can call the APIs to execute actions.

Operating System: The operating system has a Linux-based Kernel optimized for Arm architecture. The Linux Kernel is modified to support the RT-Preempt patch, enabling the Linux operating system for real-time tasks. Moreover, the patch gives the benefit of being able to completely shut down an

This work is supported by NSF CSR 1815047.

DOI 10.1109/ISPASS48437.2020.00036

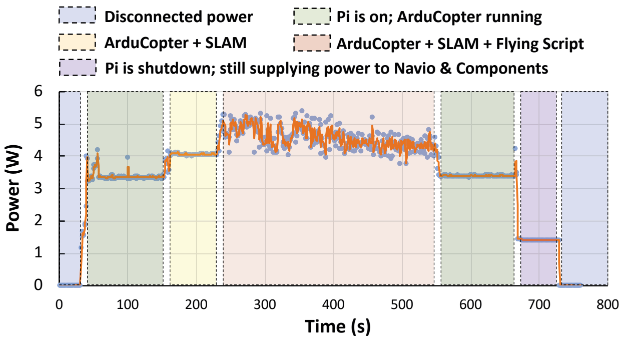

Fig. 2. Power graph of the Raspberry Pi in different conditions. In detail, first, the Raspberry Pi is not connected to a power source. Next, the power is plugged in and the Raspberry Pi boots up with ArduCopter software running (avg 3.4 W). Then, SLAM is run with an average power usage of 4 W. At 230s, the drone starts flying. The flying script is then run with an average power consumption of 4.6 W, together with SLAM. At the end, the Raspberry Pi is shutdown, only supplying power to the Navio2 HAT.

instance of a drone mission (a collection of pre-defined flight commands) and spool up a new mission while the drone is executing the current mission. Commercially available drones from DJI [8] and SkyDio [2] are capable of changing their missions mid-flight. However, they are unable to completely shut down their autopilot binary and load a different one since access to their autopilot architecture is limited. The only way to achieve this general purpose behavior in a drone is to run different autopilot binaries, each which is optimized for a certain task. This ability opens up the field of general purpose drones to the mass consumer and industrial markets. We also modified the Linux Kernel to support continuous loop-back and server instances so the drone could be controlled using multiple devices such as telemetry or SSH. The OS runs the ArduCopter binary as a multi-threaded service with high priority. Simultaneously, the OS listens on `dev/tty` for UDP packets containing a specific MavLink [9] header to indicate the packet is for the ArduCopter Service.

ArduCopter: ArduCopter is a versatile and powerful open source flight control stack which provides flying functionality. Arducopter utilizes the Navio2 HAT to interface with the motors and on-board sensors. The software utilizes the RT-Preempt Linux patch to ensure a lag-free response to any input it receives. ArduCopter takes input from the loop-back ports and receiver and transforms the corresponding action into respective PWM signals which are then controlled by the Navio2 HAT. Additionally, using the WAF compile chain [10], we are able to deploy any firmware during a mission. ArduCopter also runs multiple daemons which are able to perform the firmware switch and performance analysis.

Secondary Programs and Workloads: To characterize our drone, we simultaneously ran the ArduCopter software with ORB-SLAM [6]. ORB-SLAM is a localization and mapping algorithm that enables an agent to map an unknown environment as well as maintain its location in that environment. SLAM is very important to drones as it increases the versatility of the drone flight stack in autonomous missions. Further, characterizing the drone with the SLAM workload enables us to gauge the metrics we would expect to see if the drone was

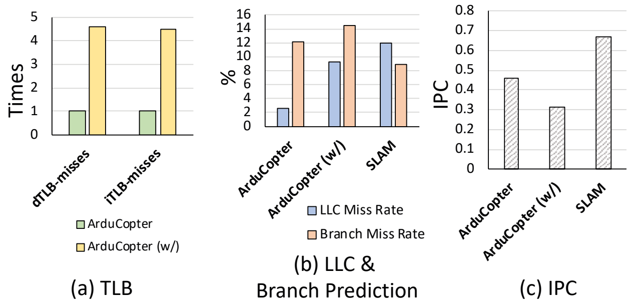

Fig. 3. Performance metric measurements. (a) ArduCopter with SLAM dTLB and iTLB misses normalized to only execute ArduCopter. (b) LLC and branch prediction miss rates for ArduCopter, ArduCopter with SLAM, and SLAM. (c) IPC for ArduCopter, ArduCopter with SLAM, and SLAM.

further developed with additional cognitive workloads.

III. Characterizations

A. Power Measurements

The power consumption of the Raspberry Pi was measured by using a USB digital multimeter which records measurements to an Excel file once every second. The results are displayed in Table 1 and Fig 2. The average power consumption of the Raspberry Pi when executing ArduCopter was measured to be 3.39 W. The average power consumption increased by 0.66 W when additionally running the SLAM workload. Finally, running the autonomous flight script on top of the current workload led to high variations in power consumption, producing an average value of 4.56 W, an increase of 0.51 W.

TABLE I
Power consumption measurements

Experiment	ArduCopter	ArduCopter +SLAM	ArduCopter +SLAM +Fly
Avg. Power Consumption	3.39 W	4.05 W	4.56 W
Peak Value	4.21 W	4.18 W	5.40 W
σ/SD	0.13	0.09	0.34

B. Performance Measurements

To measure performance, we used Linux perf and carried out analysis on the executing processes. Figure 3 illustrates our performance measurements. Figure 3a shows that when ArduCopter is running with SLAM, the TLB misses increase by 4.5x. Figure 3b show a similar trend in LLC and branch prediction miss rates. Additionally, Figure 3c sheds light on these trends. SLAM is heavily compute and memory bounded. Therefore, although ArduCopter has a higher priority, it experiences performance slowdown when SLAM is running. The impacts of these findings indicate that running additional workloads would decrease drone flight time as it leads to higher power consumption. Moreover, running such heavy workloads would increase response time from the autopilot software, underscoring the need for better optimizations.

IV. Conclusion & Future Work

We plan on using the metrics obtained to further develop the autonomous drone. We aim to continue more research into optimizing the flight-stack and drone collaboration for deep learning tasks [11], [12]. We are building a baseline model for a general-purpose drone capable of switching between firmware versions and changing missions mid-flight.

References

[1] L. P. Koh and S. A. Wich, “Dawn of drone ecology: low-cost autonomous aerial vehicles for conservation,” *Tropical Conservation Science*, vol. 5, no. 2, pp. 121–132, 2012.

[2] SkyDio, “Skydio,” skydio.com.

[3] ArduPilot, “Ardupilot,” ardupilot.org.

[4] Emlid, “Emlid navio2 hat for raspberry pi,” emlid.com/navio.

[5] S. Jijina, A. Amyette, R. Hadidi, and H. Kim, “Towards a general purpose cognitive drone,” *The Fourth Workshop on Cognitive Architectures (CogArch 2020)*, 2020.

[6] R. Mur-Artal and J. D. Tardós, “Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras,” *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[7] DroneKit, “Dronekit api,” dronekit.io.

[8] DJI, “Skydio,” dji.com.

[9] Wikipedia, “Mavlink,” wiki/MAVLink.

[10] T. Nagy, “Waf,” waf.io.

[11] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, “Towards collaborative inferencing of deep neural networks on internet of things devices,” *IEEE Internet of Things Journal*, 2020.

[12] R. Hadidi, J. Cao, M. Woodward, M. S. Ryoo, and H. Kim, “Distributed perception by collaborative robots,” *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3709–3716, 2018.