Quantifying the Design-Space Tradeoffs in Autonomous Drones

Ramyad Hadidi
Bahar Asgari
Georgia Institute of Technology
Atlanta, Georgia, USA

Sam Jijina
Adriana Amyette
Georgia Institute of Technology
Atlanta, Georgia, USA

Nima Shoghi
Hyesoon Kim
Georgia Institute of Technology
Atlanta, Georgia, USA

ABSTRACT

With fully autonomous flight capabilities coupled with user-specific applications, drones, in particular quadcopter drones, are becoming prevalent solutions in myriad commercial and research contexts. However, autonomous drones must operate within constraints and design considerations that are quite different from any other compute-based agent. At any given time, a drone must arbitrate among its limited compute, energy, and electromechanical resources. Despite huge technological advances in this area, each of these problems has been approached in isolation and drone systems design-space tradeoffs are largely unknown. To address this knowledge gap, we formalize the fundamental drone subsystems and find how computations impact this design space. We present a design-space exploration of autonomous drone systems and quantify how we can provide productive solutions. As an example, we study widely used simultaneous localization and mapping (SLAM) on various platforms and demonstrate that optimizing SLAM on FPGA is more fruitful for the drones. Finally, to address the lack of publicly available experimental drones, we release our open-source drone that is customizable across the hardware-software stack.

CCS CONCEPTS

• **Hardware** → **Analysis and design of emerging devices and systems**; • **Computer systems organization** → **Embedded and cyber-physical systems**; *Architectures.*

KEYWORDS

autonomous drones, design-space analysis, open-source platform

ACM Reference Format:
Ramyad Hadidi, Bahar Asgari, Sam Jijina, Adriana Amyette, Nima Shoghi, and Hyesoon Kim. 2021. Quantifying the Design-Space Tradeoffs in Autonomous Drones. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21), April 19–23, 2021, Virtual, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3445814.3446721

1 INTRODUCTION & MOTIVATION

Over the last decade, significant progress has been made in the development of autonomous systems. The numerous advances in drones popularized by quadcopters [1] is partly due to the countless applications addressed by these systems, such as aerial mapping [2, 3], exploration [4, 5], military [6], natural disaster recovery [7], search and rescue [8], ecology [9, 10], and entertainment [11–14]. The quadcopter design possesses many advantages over other aerial vehicle designs in terms of simplicity and efficiency [15–17]. Thus, quadcopters are becoming prevalent and many control, planning, and perception methods have been assimilated for them [15, 18–28].

Nevertheless, drones must operate under conditions that are quite different than any other compute-based agent. First, weight and power are restrictive parameters in drones. Second, drones must arbitrate between their limited compute, energy, and electromechanical resources not only based on the current tasks and local conditions (*e.g.*, wind, air temperature), but also according to the flight plan. Despite huge technological advances in drones, these problems have been approached in isolation, and the end-to-end design-space tradeoffs are largely unknown.

As a result of such isolated problem solving, architecting end-to-end drone systems and their computation landscape still remains an open question. For example (Figure 1), if we are making a special chip for drones, is it useful to improve processor performance and, if yes, is it because of energy savings or better control? How useful is improving processor power efficiency given that the majority of power consumption is coming from resources other than computing power? Should we focus on optimizing the flight-related tasks, or should we focus on secondary tasks such as recognition and autonomy? These questions pertain to creating cost-effective solutions with low system integration cost, reasonable development time, and effectiveness on drone metrics. Prior studies [29, 30] have proposed a closed-loop simulator and benchmark suite, which does not completely answer the above questions because it is focused on *high-speed* drones (more in §6).

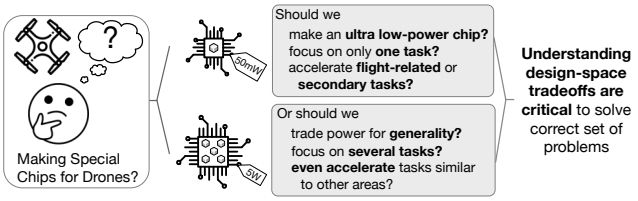


Figure 1: Impactful contributions in drones are only realized by quantifying the design-space tradeoffs.

To answer such questions and solve worthy research problems, we need to understand fundamental drone subsystems, classify drone computations and their requirements, extract design-space tradeoffs, and have access to a reproducible experimental platform.

This is the first paper to formalize and quantify the design-space tradeoffs of autonomous drone systems. To do so, first, we address

ASPLOS '21, April 19–23, 2021, Virtual, USA

ACM ISBN 978-1-4503-8317-2/21/04…$15.00
https://doi.org/10.1145/3445814.3446721

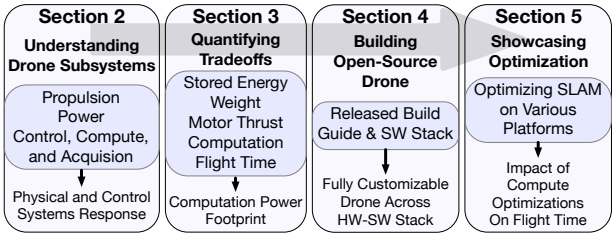


Figure 2: Sections overview.

the lack of a publicly available and reproducible experimental drone framework that is customizable across its hardware-software stack by releasing our open-source drone. Then, by exploiting this new experience, we study the computational profile and landscape of such systems, in which we must understand three major drone metrics: flight time, control response time, and autonomous features. Despite the extensive knowledge in our community, we discover several missing pieces of the puzzle in understanding conventional, described in the following.

(1) Flight time: Flight time is determined by the power consumption of the drone during flight and the battery capacity. But, the power consumption is dependent on several factors: drone weight, motor types, flying activity, and several other factors. The battery capacity also directly affects weight; a larger battery is heavier, but has a higher capacity. **(2) Control response time:** The control response time of a drone is determined by its control system. However, we do not know if additional computation power would enhance this system. **(3) Autonomous Features:** With several exciting new applications in drones, such as machine learning applications, it is important to understand how they interact with the main control system, what their computation profile is, and know how to quantify any opportunity for optimization.

To answer the aforementioned questions for flight time, after introducing fundamental propulsion and power systems (§2.1.1, §2.1.2), we extract crucial metrics from over 300 commercial components and 150 manufacturers (§3.1) to find the major relationships in determining the weight and power consumption of drones (§3.2). Using the empirical measurements and physics, our method directly translates compute power efficiency to flight time by untangling the multifaceted relationships in drones. For instance, we quantify the percentage of computation power from total power widely ranges from 2–30%, enabling gaining of up to +5 minutes flight time.

In §2.1.3, we analyze the control system of drones, namely, inner-loop and outer-loop controls. For instance, we discover that the critical inner-loop controls in drones have an update frequency of 50–500 Hz, which is not limited by computation power, but by the physical response of the drone. Finally, in §2.2, we shed light on the wide variety of autonomy in drones, current customized compute boards for drones, and discover that these systems are highly dependent on a core family of algorithms, namely, simultaneous localization and mapping (SLAM). Then, in Figure 2 in §3, we present several important tradeoffs in drones, including the computation power footprint. Next, in §4, we develop our open-source platform. Finally, in §5, we showcase optimizations for SLAM on various platforms. For instance, we show that moving from GPU/CPU to FPGA provides 20x power savings, enabling 15–20% (+2–3 minutes) of additional flight time in small drones.

This is the *first paper* to contribute the following:

- Formalizes the fundamental drone subsystems and quantifies the design-space tradeoffs for the computational profile of drones to discover how computation power consumption affects drone flight time, accomplished by incorporating physics and empirical measurements from 300 commercial components and 150 manufacturers.
- Clearly separates the required computing for inner-loop controls (real-time requirements) *vs.* outer-loop controls (autonomous features) in drones and outlines the required computation amount and benefits gained.
- Showcases the optimization landscape of the widely used SLAM algorithm in autonomous drones and the effects on flight time by using the presented data.
- Develops an open-source and reproducible platform with a customizable hardware-software stack to address the lack of publicly available drone platforms.

2 AUTONOMOUS DRONES

Autonomous drone subsystems determine several crucial properties of a drone, and the associated design choices have a pivotal impact on the effectiveness of the end-to-end system. However, each subsystem has been studied in isolation. This section first briefly introduces these subsystems, and then extracts necessary details pertaining to computations.

2.1 Fundamental Subsystems

Figure 3 overviews the main subsystems for a quadcopter drone. We divide the fundamental subsystems as follows: propulsion system, which generates necessary force for movement and lift; power delivery system, which delivers the power to electromechanical components; and control, compute, and acquisition system, which controls and stabilizes the drone with the help of sensors.

2.1.1 Propulsion System. Quadcopter drones utilize four identical motor-rotors, two pairs of which spin in opposite directions, for the generation of thrust (*i.e.*, uplifting force). For maneuvers, drones must precisely change the rotation speed of each rotor, which along with their small size, necessitates electrical propulsion with batteries. Thus, only direct current (DC) motors, specifically brushless DC (BLDC) motors, are used. BLDC motors achieve higher rotation speeds with improved control, while providing precise feedback for measuring rotation speed. Nevertheless, BLDC motors require complex and expensive electronic speed controllers (ESCs) (§2.1.2).

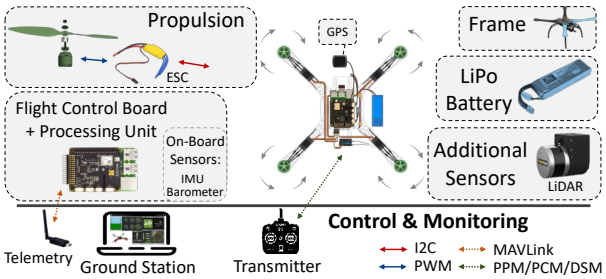


Figure 3: General overview of an autonomous drone.

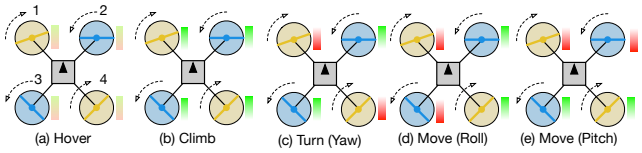


Figure 4: Essential drone movements. Colored rectangles near each rotor show how rotation speed changes, with red representing decrease and green representing increase.

The physics behind the essential movements of quadcopters is relatively simple, shown in Figure 4. By ignoring several complexities, covered in §2.1.3, all movements stem from the precise control over each rotor's thrust while accounting for several environmental factors such as wind and air density. Drones use the same uplift thrust for horizontal movements by tilting. The maximum horizontal speed depends on the maximum stable angle of attack (*i.e.*, tilt angle), which depends on the thrust-to-weight ratio (§2.3).

2.1.2 Power Delivery System. Lithium polymer (LiPo) batteries (lithium-ion polymer), which have the highest energy density (Wh/Kg) and discharge rate (measures how fast the battery can be safely discharged) in rechargeable lithium-ion technology, are the main source of power in drones. Since BLDC motors require high current, the high current flow of LiPo batteries is a critical factor. However, the downfall is that LiPo batteries are relatively fragile; only 85% ($LiPo_{DrainLimit}$) of their capacity should be used during a flight. LiPo batteries have various configurations that are multiples of cells with a nominal voltage of 3.7V/cell, studied in §3.1.

Each BLDC motor requires three-phase currents, which are generated by a separate ESC using DC current. The complexity of the ESC circuits is evident, as they need a switching frequency of 60–600 KHz while delivering hundreds of Watts. ESCs also provide necessary electronics to implement feedback to achieve precise control of the rotation speed of their own microcontroller. ESC protocols usually go beyond PWM (pulse-width modulation) signals for modern-day drones due to high precision in control (*e.g.*, the DShot1200 protocol has a communication frequency of 74.6 KHz). The above criteria make ESCs one of the heavier components (§3.1).

2.1.3 Control, Compute, and Acquisition System.

A. Inner *vs.* Outer Loops: The recent advancements in autonomous drone systems have mainly been accomplished with the developments of high-level algorithms in state estimation, trajectory tracking, localization, and deep learning [18, 20, 21, 24–26, 28]. Nonetheless, such high-level algorithms (*i.e.*, outer-loop computations) rely on and are directly impacted by the inner-loop control [15, 17, 22, 23] (Figure 6). High-level algorithms only provide state targets, grouped into position, velocity, and attitude[1], to the inner-loop control. The inner loop reaches those target set points over time by direct manipulation of the drone actuators while also maintaining a stable flight. Furthermore, remote controller (RC) commands and safety override commands pass through the inner-loop to minimize response latency. Table 1 summarizes a handful of dynamic effects, such as stabilization, that are compensated by the inner-loop control for a stable flight, emphasizing inner-loop control relative importance to the high-level algorithms.

[1]Defined as orientation of a solid body around three Cartesian axes.

Table 1: Unpredictable effects compensated by the inner-loop control *vs.* decisions made by the outer-loop control.

	Inner-Loop Control		Outer-Loop Control	
Stabilization	Wind gusts	Set Target	Position target	
	Local disturbance		Attitude target	
	Atmospheric turbulences		Velocity target	
	In-door & close-to-object		Navigation & trajectory	
	Propeller flapping		Obstacle Detection*	
Control	Translational lift/thrust		Planning	
	Absolute speed	Applications	SLAM*	
	Weight imbalance		LiDAR Mapping	
	Motor imperfection		Sonar Mapping	
	Angle of attack		...	
	Flight time			
	ESCs management			

* Some implementations are across inner and outer loops

B. Hardware-Software Stack: Figure 5 shows the hardware-software stack abstraction of a drone. The flight controller boards with additional on-board sensors directly manipulate ESCs and sensors. Flight controllers have the following main components (Table 4 provides some examples): (i) a microcontroller (MCU) usually STM32F 32-bit Arm Cortex-M series; (ii) one or two 6-axis inertial measurement units (IMUs); (iii) a barometer, for altitude measurements; (iv) and possibly several chips for sensors, video feed codec, and communication. If necessary, external sensors with their dedicated full-stack supporting system are added. The operating system (OS) is dominated by Linux, except for racing applications. The hardware abstraction layer (HAL) provides necessary APIs. The shared libraries layer provides common sensor fusion algorithms (*e.g.*, Extended Kalman Filter). The control layer is described in the next paragraph. The final application-specific flight code layer largely depends on the application. Finally, the communication layer delivers stats to the ground station and, if necessary, a MAVLink [31] protocol offloads computations to another node.

C. Inner-Loop Control: In the inner loop, the control layer uses the *on-board* sensors to stabilize the drone and reach to target set points dedicated by the outer loop. This layer extensively uses high-performance hierarchical proportional-integral-derivative (PID) controllers, whose filter response and quality of the estimated state variables defines the drone behavior. The feedback loop is shown in Figure 6 and is completed by sensor measurements. The control is performed hierarchically[2] by dividing the control problem into three levels depending on their response time, shown in Table 2b, known as time scale separation. The three levels are as follows: High-level position or trajectory, mid-level attitude, and low-level thrust controller [15, 16, 23, 32]. *Based on Table 2, no higher update*

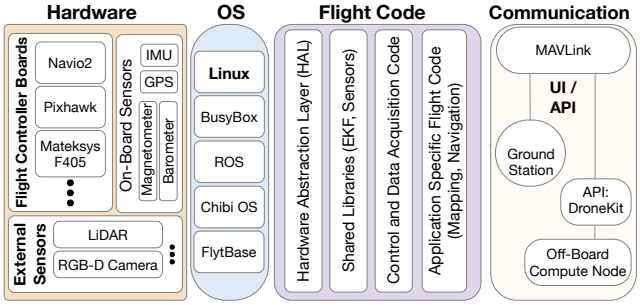
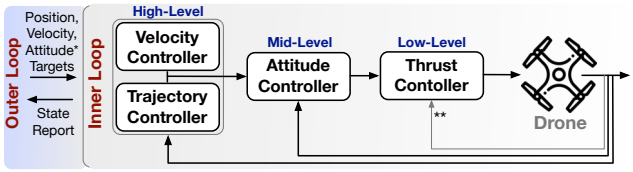


Figure 5: Hardware-software stack abstraction of a drone.



*If the application requires attitude control by the outer-loop; otherwise, the control happens entirely in the inner-loop. ** This feedback loop is usually implemented through attitude control because of its slow response time

Figure 6: Inner- and outer-loop controls in drones. The inner-loop control manipulates the drone actuators to reach the target set points dictated by the outer-loop control.

Table 2: (a) Common data frequencies of on-board sensors. (b) Update frequency and response time of each controller.

Sensor	Data Frequency
Accelerometer	100–200Hz
Gyroscope	100–200Hz
Magnetometer	10Hz
GPS	10—20Hz
Barometer	1—40Hz

(a) Sensors Data Frequency

Controller	Update Frequency	Response Time
Thrust	1KHz	50ms
Attitude	200Hz	100ms
Position	40Hz	1s
Inner-Loop	50—500Hz	N/A

(b) Controllers Update Frequency

and response frequency than 1 KHz is necessary, both for reading the sensors and updating the controllers.

D. Inner-Loop Control Computations: We summarized the inner-loop control computation as two groups: (i) filter computations such as EKF for data fusion and updating PIDs, and (ii) algebraic functions for state estimation such as air drag and trajectory. The filter computations consist of keeping a history and accumulated versions of previously observed measurements, their derivative, and their integral. The state estimation includes 3x3 matrix operations based on the measurable state of the drone that includes $x = (\vec{\zeta}, \dot{\vec{\zeta}}, \vec{\Omega}, R)$, in which $\vec{\zeta} \in \mathbb{R}^3$ is the position (using data from the IMU, GPS, and barometer), $\dot{\vec{\zeta}}$ is the velocity (using IMU), $\vec{\Omega} \in \mathbb{R}^3$ is the angular velocity (using IMU), and $R \in SO(3)$ is the attitude (using IMU) of the drone [17, 32]. All control computations are effectively performed by a STM32F 32-bit Arm Cortex-M, a single-core processor with a frequency on the order of 100 MHz, in even high-speed racing drones. Although some research proposals suggest replacing control-theory-based with learning-based algorithms that require higher computation capabilities [27], the consensus is that unless a new electromechanical part introduces a drastically new response time, higher computation capabilities are not required. For instance, the inner-loop update frequency in high-end commercial products [33, 34] ranges from 50 Hz to 500 Hz. Even for highly specialized sensor-based control techniques with incremental nonlinear dynamic inversion (INDI) that can stabilize a drone under powerful wind gusts [22], the update frequency is still 500 Hz. *Thus, the update frequency of the inner-loop control is 50–500 Hz, which is limited by the physical response time and inertia of the control and electromechanical components in drones and is not limited by the computation power.*

[2]Hierarchical controllers are non-linear controllers that yield stability and enhanced robustness, especially for highly nonlinear dynamics (*e.g.*, air drag). Other linear and non-linear controllers for drones [16] also have a similar update frequencies.

2.2 Autonomy in Drones

Autonomy in drones is realized by intelligently providing target states (*i.e.*, position, velocity, and sometimes attitude) with the computation that occurs in the outer-loop control, as explained in §2.1.3. Although autonomy is a defined term in self-driving vehicles, meaning to safely navigating from point A to B, such a definition is not set in drones. For instance, mapping drones are autonomous in the sense that they fly within a predefined airspace while covering the entire area for mapping [3, 5, 10, 35]. Or, active-filming drones use vision cameras and recognition technologies to follow a predefined target and optimize the filming angles while avoiding obstacles [1, 11, 33]. As a result, autonomy in drones is still an active area of research and commercial products.

The outer-loop computations always occur in isolation, and the hardware dedicated for such computations varies from quad-core and Intel i7 CPUs in research-oriented studies [5, 21] to custom-built computers based on NVidia Jetson TX2 [33, 36] or custom Intel boards (*e.g.*, Intel Aero compute board [37]) in commercial products. *From a computation perspective, to ensure that the inner-loop control is in real time, the computations for autonomous tasks in the outer-loop are not co-located on the same computation core or even the same unit as for the inner-loop control.*

We find a wide variety of high-level algorithms in autonomous drones are dependent on a core family of algorithms, namely, simultaneous localization and mapping (SLAM) and visual odometry (VO) [38–42]. These algorithms are the fundamental building blocks for many autonomous technologies [19, 39, 43] and are used in various tasks such as navigation, obstacle avoidance, and path planning. Designing drone systems that provide accurate localization in real-time on platforms with limited computational and energy resources is an active area of research [18, 19, 24]. Therefore, to date, various implementations of SLAM with the focus on algorithmic-level optimizations [38, 40–42, 44] or hardware acceleration [43, 45–50] have been proposed. We explore such hardware optimizations in §5.

2.3 Drone Design Metrics

Table 3 presents the the definition of metrics used in drones. Most of the metrics are not standalone and are dependent on each other based on the design choices covered in §3.

3 QUANTIFYING DESIGN-SPACE TRADEOFFS

To understand the computational profile in autonomous drones, we must quantify the design-space tradeoffs that define several important features such as flight time. This section quantifies each tradeoff correlation with weight, which defines the order of power consumption in a drone. Then, we derive how the computation affects this design space.

3.1 Important Tradoffs

Battery Stored Energy & Weight. Drones predominantly use LiPo batteries (§2.1.2), which constitutes a large fraction of a drone's weight. Although a higher capacity battery has more charge, the additional weight may result in a shorter flight time (not to mention the additional weight of sensors and computation units). Hence, it is crucial to understand the relationship between the capacity (mAh) and the weight of the batteries. Although the range of energy

Table 3: Definition of important metrics.

Metric	Definition
Thrust-to-Weight Ratio (TWR)	The maximum total thrust produced by the motors (g) divided by the drone's weight (g). Common ratios are from 2:1 to 7:1. A higher ratio enables drone to perform elaborate aerobatics. Higher ratios mean heavier motors, larger batteries, and ESCs with a higher consumption. To find the highest possible contribution boundary of computation power consumption, we use a TWR of 2:1.
Thrust Per Motor	Thrust that is produced by a motor depends on the propeller diameter and pitch, supply voltage, K_v rating, and motor design. K_v rating is used to calculate the rotation speed (RPM), ω, of the motor per supply voltage, $\omega = K_v \times V$. So, for a fixed voltage, a lower K_v rating produces more torque and turns larger propellers. A propeller with a larger diameter and pitch moves a larger volume of air per rotor revolution and provides larger thrust. The maximum propeller size is determined by the frame size, or wheelbase.
Discharge Rate	The battery discharge rate or C rating is a measurement of the maximum continuous current a battery can safely supply. The maximum continuous current from C rating is calculated as Capacity(Ah) $\times C$ = I.
Battery (xSyP)	A LiPo battery has a nominal voltage of 3.7V/cell. To supply more voltage, cells are packed together in series. The convention is to write the configuration as xPyS, which means x pack of y cells in series. To provide a high thrust-to-weight ratio, we need motors with a lower K_v rating for higher torque, which means a higher voltage is required to achieve good RPMs for lifting.
ESC Max. Current	ESCs must be able to supply constant current to the motors while the drone is flying. The maximum continuous current value shows how much current an ESC can handle, which directly depends on the type of motor and propeller.
Frame Wheelbase	The frame size or wheelbase is the distance between two diagonal arms of a quadcopter. The wheelbase defines the maximum propeller diameter a drone can use. Indoor drones have a wheelbase under 100 mm, while outdoor drones have wheelbases up to 1000 mm.

densities of LiPo batteries are known, these values are insufficient for accurate estimations for two reasons. First, we are interested in the end product, which also includes casings, wires, and protection circuits. Second, as the manufacturing process is not ideal with various discharge rates, estimation based on energy density is not precise. To address this knowledge gap, we study 250 commercial batteries. By grouping the batteries based on their configuration in number of cells (see §2.3), we derived a linear relationship between the capacity and the weight of the batteries, shown in Figure 7. Generally, for batteries with higher voltages (for motors with higher torque), we observe a higher overhead. However, these batteries are necessary to lift the drone. The figure also includes discharge rates, which result in heavier batteries, but the resulting weight does not deviate from the extracted formulas per battery configuration.

ESCs Current & Weight. ESCs provide high continuous current with a high switching frequency (*i.e.*, $6 \times \text{RPM}_{\text{rotor}}$), both of which are determined by the motor and TWR ratio. ESCs have large MOSFETs with high source-to-drain voltage and very high continuous current. Therefore, the weight of ESCs are highly correlated with the maximum continuous current they can handle. We extract this correlation by studying 40 commercial ESCs, shown in Figure 8a. We divide the ESCs into two groups: Short-flight (under 5 minutes)

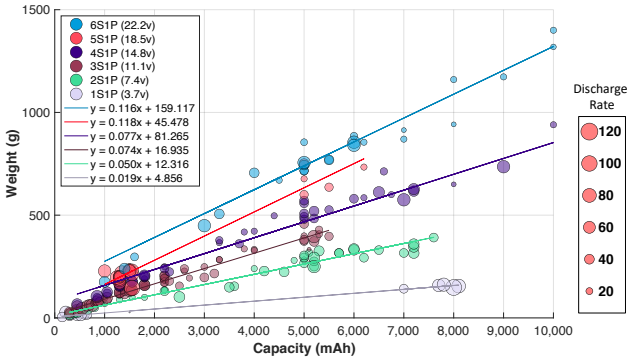


Figure 7: LiPo battery capacity and weight per configuration (§2.3), extracted from 250 commercial batteries.

ESCs, targeting racing drones; and long-flight ESCs, targeting all other use cases. In racing, ESCs are designed with lighter MOSFETs and capacitors that overheat in longer flights.

Frames. A larger drone frame size leads to more choices in the components, ability to house new sensors, and larger propellers. However, even with carbon and glass fiber technology, the weight of a frame is not negligible. Thus, we study 25 commercially available frames in Figure 8b and extract the correlation between their weight and wheelbase.

Propulsion System. The motors and propellers of drones have a wide variety of configurations; thus, the tradeoffs of the propulsion system are multifaceted and complex. The main deciding factor in the process is the target TWR. Since we are interested in understanding the computational profile in the most efficient designs, we set the target TWR to 2, the minimum required value for flying. Thus, the derived values specify the highest percentage of possible contribution of computation power. Figure 9 shows an extrapolated relationship between the max current draw of the appropriate motors and the corresponding drone's basic weight (*i.e.*, not including battery, ESCs, and motor weight) grouped by the supply voltage (*i.e.*, the cells of the LiPo battery). For each frame, we first set the maximum propeller diameter in inches dictated by the wheelbase (written in the legend). Then, we extract the thrust and K_v rating of the motors from data released by 150 manufacturers. Then, by varying the weight and supply voltage, we calculate the minimum required max current draw per motor.

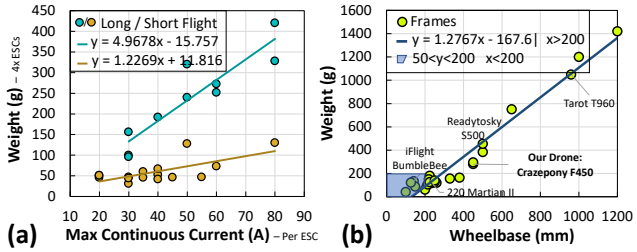


Figure 8: (a) The maximum continuous current per ESC and the total weight of ESCs, extracted from 40 ESCs. (b) Frame wheelbase and its weight, extracted from 25 frames.

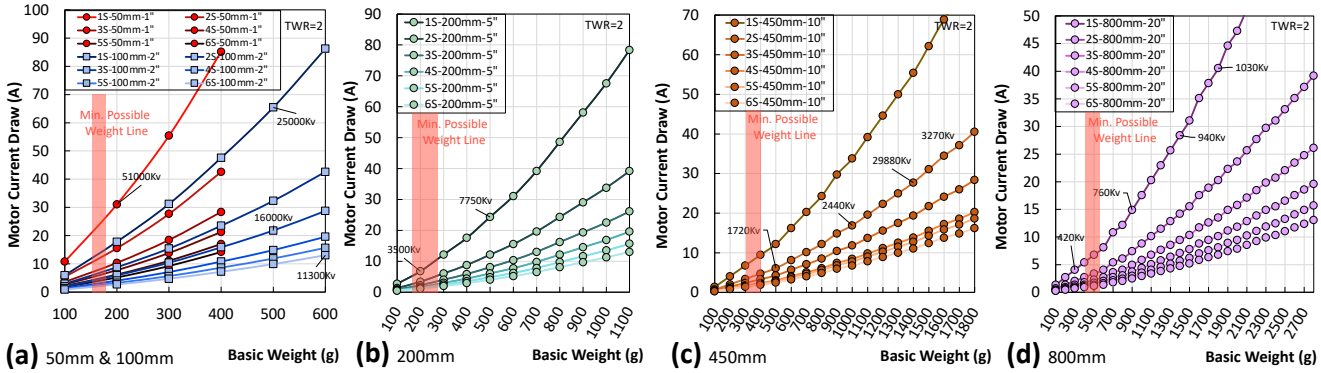


Figure 9: Relationship between the max per-motor current draw and the basic weight, grouped by supply voltage and wheelbase sizes from 50mm–800mm. TWR is 2 and data is extracted from 150 manufacturers.

Table 4: Specifications of common flight controllers, compute boards, and external sensors for drones.

	Name	Weight (g)	Power
Flight Controllers & Computation			
Basic	iFlight SucceX-E F4 [56]	7.6	<100mA@5V
	DJI NAZA-M Lite [57]	66.3	300mA@5V
	DJI NAZA-M V2 [58]	82	300mA@5V
	Pixhawk 4 [59]	15.8	400mA@5V
	MateksysF405 [60]	17	200mA@5V
Improved	Intel Aero [37]	30	2A@5V
	Navio2 [61]	23	150mA@5V
	Raspberry Pi 4 [62]	50	1A@5V
	Nvidia Jetson TX2 [63]	85	2A@5V
	DJI Manifold [36]	200	20W
External Sensors			
FPV	Eachine Bat 19S 800TVL	8	50mA@5V
	RunCam Night Eagle 2 [64]	14.5	200mA@5V
LiDAR	HoverMap [65]	1800	50W, Self-Powered
	YellowScan Surveyor [66]	1600	15W, Self-Powered
	Ultra Puck [67]	925	10W, Self-Powered

In Figure 9, we see that heavier drones have motors with higher K_v ratings for higher rotation speeds. Moreover, in larger wheelbases, larger propellers are needed to lift the drones. This is because it is physically impossible to use smaller propellers with high RPMs. These large propellers require higher torque from the motors. Thus, these motors have a lower K_v rating (compare K_v ratings in Figure 9a *vs.* b). However, because of their larger size (to create the necessary torque, the motors have a greater number of poles and larger diameters), these motors are much heavier (*e.g.*, from 5 g/motor in 100 mm drones to 100 g/motor in 1000 mm drones).

Flight Controllers, On-Board Computation, & Sensors. Table 4 lists common open-source and commercial flight controllers, additional computation boards, and external sensors. All of the flight controllers have an integrated STM32F Arm Cortex-M processor series as the main inner-loop controller (§2.1.3). We divide the flight controllers into two groups: basic, which provides only necessary inner-loop functions with limited outer-loop capabilities; and improved, which provides customizable inner-loop functions and a few outer-loop functions. In commercial markets [33, 36, 37], the Nvidia Jetson TX2 embedded board is considered a high-end solution with a price of $300. The power consumption of these compute boards ranges from 0.5–20 W. Therefore, in the following section, we assumed two levels of power consumption: a 3 W and a 20 W chip, representing basic and advanced flight controllers, respectively. For external sensors, we list the first-person view (FPV) cameras with a maximum of 1 W consumption. High-definition (HD) cameras are self-powered with weights around 100 g. Specific LiDAR solutions optimized for drone technologies are also listed in the table for completeness. All options are stand-alone and weigh around 1 kg. To make integration easier, state-of-the-art LiDAR solutions have their own battery and compute boards. We study how the addition of these sensors due to their weight, reduces the contribution boundary of main computation power in large drones.

3.2 Computation Footprint

Procedure: To understand the computational profile, we derived the total power consumption of a wide range of drones from small indoor drones (100 mm wheelbase) to large military and filming drones (800 mm wheelbase). We use §3.1 extracted data while accounting for the additional weight and power consumption of each module. In detail, per each frame (Figure 8b), we choose the propeller with the maximum size, find the required RPM for the motors, and choose the best matching motor depending on the number of cells in the LiPo battery, while sweeping the range in the capacity of the batteries from 1000mAh to 8000mAh (Figure 7, Equation 4). Then, from the maximum motor current draw (Figure 9, Equation 2), we choose ESCs (Figure 8a). In this step, if the additional weights necessitate a new motor, we redo the previous steps (Equation 1). By assuming a low-load hovering condition ($\text{Flying}_{\text{Load}}$, 20–30% of the maximum current draw) with 85% LiPo battery capacity limit ($\text{LiPo}_{\text{DrainLimit}}$), we calculated the power consumption (Equation 7), shown in Figures 10a,b, and c for 100, 450, and 800 mm wheelbases.

$$\text{Weight}_{\text{Total}} = \mathcal{F}(4W_{\text{Motor}}, W_{\text{ESC}}, W_{\text{Battery}}, W_{\text{Frame}}, W_{\text{Propellers}}, W_{\text{Compute}}, W_{\text{Sensors}}, W_{\text{Wires}}) \quad (1)$$

$$\text{Motor}_{\text{Current}} = \mathcal{G}(\text{Weight}_{\text{Total}}, \text{TWR}) \quad (2)$$

$$\text{Power}_{\text{Avg}} = \mathcal{H}(\text{Motor}_{\text{Current}}.\text{Batt}_{\text{V}}, \%\text{Flying}_{\text{Load}}, \text{Power}_{\text{Compute}}, \text{Power}_{\text{Sensors}}) \quad (3)$$

$$\text{Batt}_{\text{Capacity}} = \mathcal{M}(\text{LiPo}_{\text{Capacity}}, \%\text{Power}_{\text{Eff}}, \%\text{LiPo}_{\text{DrainLimit}}) \quad (4)$$

$$\text{FlightTime} = \mathcal{N}(\text{Batt}_{\text{Capacity}}, \text{Power}_{\text{Avg}}) \quad (5)$$

$$\%\text{Power}_{\text{Computation}} = \mathcal{X}(\text{Power}_{\text{Avg}}, \text{Power}_{\text{Compute}}) \quad (6)$$

$$+\text{FlightTime}_{\text{Compute}} = \mathcal{Z}(\%\text{Power}_{\text{Computation}}, \text{FlightTime}) \quad (7)$$

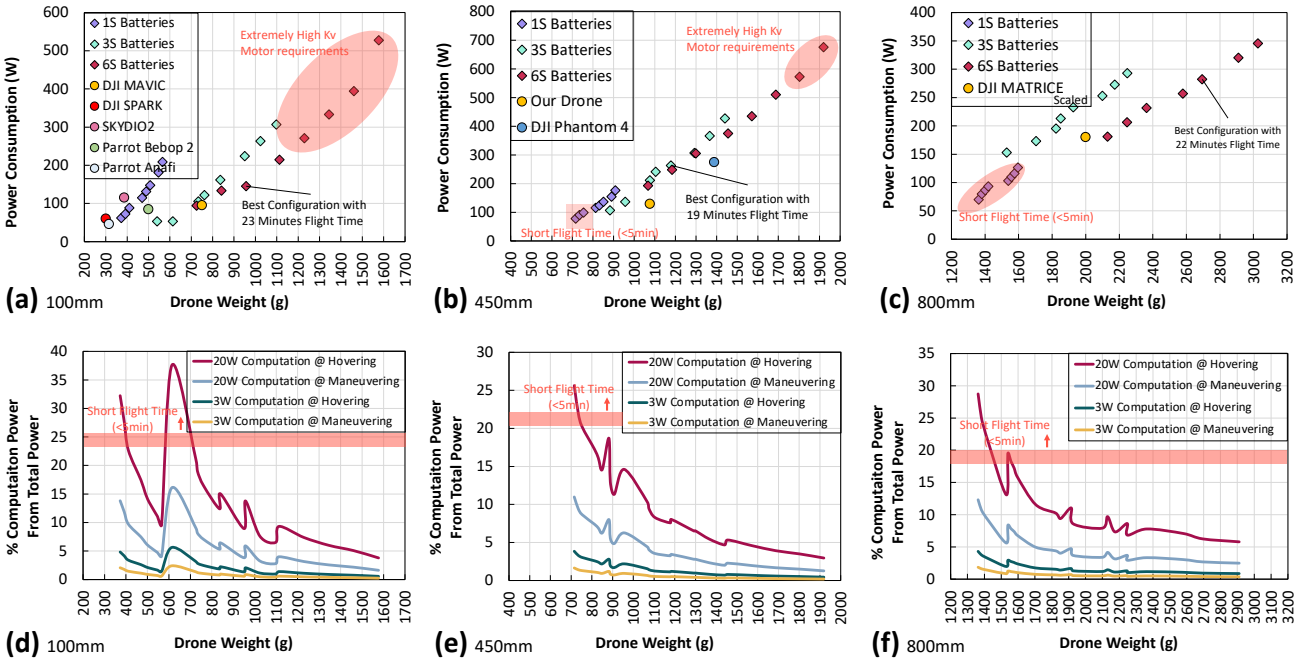


Figure 10: Top Row (a,b,c): The total power consumption of drones with various wheelbases extracted by relationships in Equation 3.2 and verified with data from commercial drones shown as additional data points [33, 51–55]. Bottom Row (d,e,f): The computation footprint considering 3 W and 20 W chips shown with 3/20W computation @ hovering/maneuvering lines.

Validation: We validate our data by adding commercial drone data points using the released flight times and battery configurations [33, 51–55], shown as additional diamond-shaped data points in Figure 10. No data skewing or pre-selection is used for extracting tradeoffs (*i.e.*, all data points are shown). Additionally, we verify the average power consumption by calculating the total flight time to match with current state-of-the-art commercial drones, resulting in 23, 19, and 21 minutes of flight time for 100, 450, and 800 mm wheelbases, respectively.

Interpretation: Figures 10d,e, and f illustrate the percentage of computation power from the total power of a drone in two groups with hovering and maneuvering (20–30% and 60–70% of the maximum current draw, respectively). The first group with a 3 W compute power represents a commercial ultra-low-power flight controller. The second group with a 20 W compute power represents a GPU-CPU system with much higher capabilities. First, we see that the 3 W chips have less than 5% contribution in total power consumption. Second, even for the 20 W system, when the drone moves, the contribution drops to an average of 10%. Moreover, we see jumps that occur because heavier drones need batteries with more cells to provide higher voltage for higher KV motors. However, initially, those batteries are less efficient than the batteries used for lighter drones. Also note that these drones have a target TWR of two; hence, the contribution shown is at its highest. To quantify, we can convert this power savings to extra gained flight time (see Equation 7. *In large- to medium-sized drones, the average computation power is 10% of the total power and the maximum gain of computation power savings is with +2 minutes in total flight time and possibly less considering maneuvering and higher TWR values.*

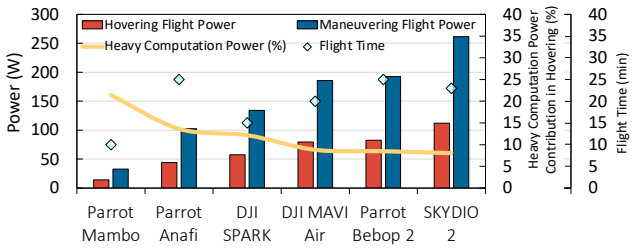


Figure 11: Commercial small-sized drones' heavy computation power contribution and their flight time.

For small-sized drones, the tradeoff between the computation and flight power is more critical. In addition to Figures 10a and d, we also study the power consumption of nano and micro commercial drones' power consumption, outlined in Figure 11 [33, 51, 53, 55, 68, 69]. For these drones, when hovering, the power consumption is from 2–7%. Nevertheless, when hovering with heavy computations (*e.g.*, face recognition, HD video recording), the contribution of computations in total power consumption reaches 10–20% (shown with a yellow line in Figure 11). *Thus, in small drones, by optimizing heavy computations such as SLAM and deep learning workloads, we can potentially increase the flight time by up to 20%, or around +5 minutes in total flight time.*

How to Use This Data: Figure 12 illustrates the procedure for how to obtain the total and compute power consumption of a drone depending on its size and battery capacity. Thus, we can understand how power savings or special chips affects the flight times and weights for all drones. We showcase SLAM in §5 as an example.

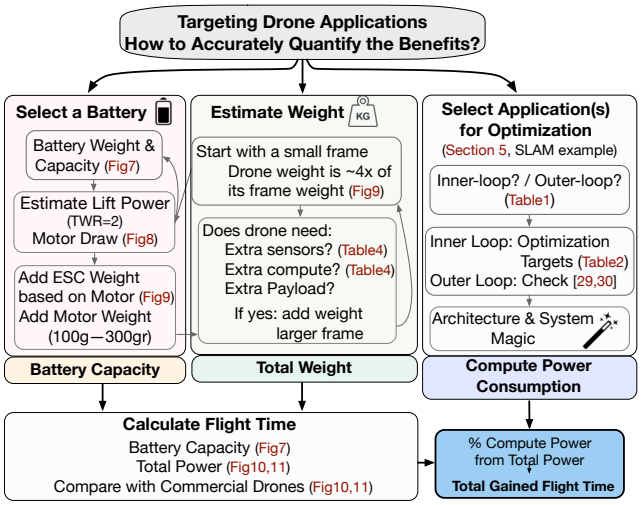


Figure 12: Procedure of quantifying total/compute power consumption in drones.

4 OUR OPEN-SOURCE DRONE

To address the lack of publicly available end-to-end experimental and reproducible frameworks for drones, we develop and fly test a fully open-source experimental drone that is fully customizable across its hardware-software stack. We integrate several widely-used hardware/software components. This platform reduces the barriers to entry to drone research[3], shown in Figure 13, and has a total cost of $500 with the ability to carry 200g additional payloads. The components, as far as they are compatible (*e.g.*, voltage, connections), can be easily switched/added to the drone. The current available alternatives do not provide full access to the hardware-software stack and have no extra-weight capacity.

Figure 13: Our Drone.

Hardware-Software Ecosystem. With the backing of the Navio2 [61] for crucial flight inner-loop control, our drone uses a Raspberry Pi[62] (RPi) with a maximum power of 5W. We integrate high-level autonomous flying firmware [70] to run advanced waypoint navigation algorithms and autonomously execute certain actions based on the results of the SLAM algorithm [71]. The following sections overview the four layers of the drone's ecosystem.

(1) High-Level Functions: The high-level functions layer consists of high-level and low-level APIs which are used to write custom code and firmware. The custom firmware is converted to a Linux service and run on the Pi in the background. We also utilized the DroneKit [72] C++ and Python APIs, which were modified to allow the drone to be reconfigured mid-flight. DroneKit allows us to connect to the drone, issue flight commands, and monitor the drone. Apart from being open-source, DroneKit is easily extensible and provides the flexibility to be used on on-board computers as well as ground-station applications by abstracting away physical MAVLink [31] protocols.

(2) Autopilot: ArduCopter[70] is an open-source autopilot code-base for drones with great versatility. ArduCopter, written in C++, allows for manual flying/autonomous control. Our modified Linux kernel allows ArduCopter to utilize loop-back ports to listen to commands being issued by external applications executing on other computers (*e.g.*, RPi). The ArduCopter binary, once compiled with WAF [73], runs several Linux daemons[74] with distributed roles.

(3) Modified Linux Kernel: The Linux kernel is modified to support the Preempt_RT patch, which enables the Linux operating system to become suitable for drones. Using this, we can completely shut down an instance of a drone mission and spool up a new mission while the drone is in mid-flight, safely and securely using WAF [73]. The Linux kernel is also modified to support continuous loop-back and server instances so the drone can be controlled using multiple devices such as through 915 Mhz telemetry or a laptop through Secure Shell (SSH).

(4) Flight Controller: We use the Navio2 controller with a Cortex-M3 co-processor, GPS, and 2x IMUs. Navio2 has generic GPIO pins for any compute board and provides connection to our RPi. During flight, the RPi sends signals to the board that are decoded by the controller.

(5) Hardware Control Surfaces: The controllable hardware consists of sensors, four motors, and ESCs. The weight breakdown of our drone is shown in Figure 14, which shows similar trends as shown in §3.1. The frame, battery, motors, and ESCs are the major components contributing to the weight.

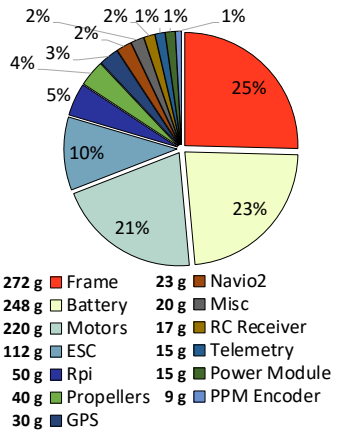


Figure 14: Our drone weight breakdown.

A New Platform Different From Current Platforms. Several popular commercial drones such as the CrazyFlie [75] or the PlutoX [76] have drastic tradeoffs between performance and flight time while limiting user access to flight code or being unable to carry additional payloads. Moreover, they can be configured only for limited purposes. With our drone, our goal is to minimize that tradeoff and give users the power to import *both* high- and low-level (*i.e.*, inner- and outer-loop) functions. Our drone can be configured for a variety of research purposes because the hardware stack is configurable. Moreover, we use Linux with the RT-Preempt patch to allow for a wide range of applications while enabling the control of the drone and parameters in real time. We choose the Navio2 flight controller because it is easily configurable for different applications and grants complete access to all control systems.

[3]The ACM artifact link on the first page or doi.org/10.5281/zenodo.4546174 for the most recent version. This open-source repository includes guide to build the drone, data sources of §3, and software stack of the drone.

5 SHOWCASING OPTIMIZATIONS

This section exhibits the impacts of design optimizations on performance and power consumption and concludes with the impact of optimization on flight time. To study this, we explore offloading ORB SLAM onto various hardware platforms.

Experimental Setup & Platforms. Our baseline platform to execute autopilot and SLAM (ORB SLAM [71]) is a RPi4 [62]. We measure the power consumption of the RPi using a USB digital multimeter that records measurements once every half second (± 10 mW). The power consumption of the entire drone is measured with a digital oscilloscope by measuring both current and voltage every 20 ms (± 0.5 mW) of the battery while controlling the drone. To measure performance at the instruction level, we used Linux perf and carried out analysis while the entire software-hardware stack is in loop. Our hardware platforms for implementing SLAM include a separate RPi4 [62], Nvidia Jetson TX2 [63, 71, 77], and a ZYNQ XC7Z020 FPGA on a PYNQ-Z1 embedded board. All the SLAM experiments run with the relevant EuRoC micro aerial vehicle dataset [78], while confirming SLAM key metrics.

For FPGA implementation, we use Xilinx Vivado HLS and describe our tailored microarchitecture in C++ by using relevant *#pragrma*. We use the post-implementation resource utilization, power consumption, and latency reported by Vivado. Inputs and outputs of the accelerator are transferred through the AXI stream interface. The clock frequency is set to 100 MHz. Similarly, we use the EuRoC dataset. For ASIC comparisons, we use the 20 mm^2 Suleiman et al. implementation on ASIC, in 65nm CMOS [19]. Navion is a visual-inertial odometry (VIO) accelerator that does not include the full-loop feedback of SLAM; nevertheless, it offers the order of power consumption in ASIC implementations. Navion processes the EuRoC dataset in real-time at 20 frames per second (FPS) while consuming a maximum of 24 mW.

5.1 Running Autopilot and SLAM on RPi

Performance. When running SLAM along with the autopilot on an RPi, SLAM in not only not fast enough, but also it negatively impacts the performance of the autopilot. For instance, the presence of SLAM causes 4.5× as many TLB misses as the autopilot alone causes. Similarly, we observed that the LLC and branch-prediction miss rates of the autopilot with SLAM are also higher than those when running the autopilot solely, as the primary axis in Figure 15 shows. Additionally, as the secondary axis in Figure 15 shows, the IPC of the autopilot decreases by 1.7×. These observations indicate that by running a few additional workloads, specifically heavy ones, the real-time response of the autopilot will lag and we will miss several outer-loop deadlines. Although the outer-loop control is not directly related to the control system, improving the performance of processors is necessary to handle heavy computations that are introduced by new workloads.

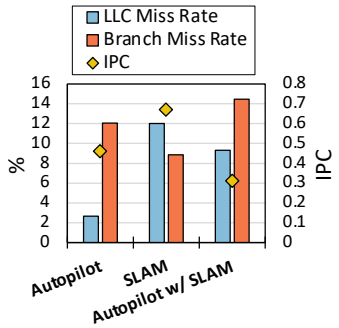


Figure 15: Performance metrics for SLAM and autopilot on RPi.

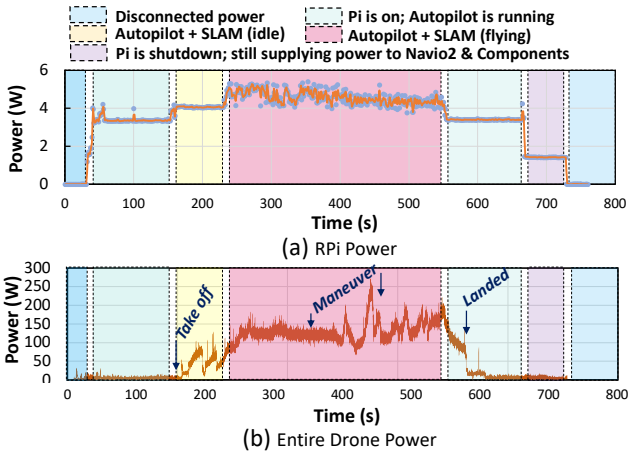


Figure 16: Power consumption graph of (a) RPi executing SLAM and autopilot code and (b) entire drone during flight.

Power Consumption. Figure 16a shows the power consumption graph of the RPi during flight. We measure the power consumption of the RPi while it is executing the autopilot software, SLAM, and flight script (*i.e.*, pre-set commands for autopilot). The average power consumption of the RPi when executing the autopilot is 3.39 W, which increases to 4.05 W when we start SLAM, but the drone is not flying yet (SLAM is idle). Finally, when the drone flies and SLAM actively processes input data, up to 5 W of power is consumed and the average power consumption of RPi reaches 4.56 W – we use these numbers to estimate heavy computation power consumption in Figure 11. Thus, by offloading SLAM onto a low-power platform such as ASIC/FPGA, we can potentially save up to 2 W, which would have a high impact for small drones (*e.g.*, Parrot Mambo [68]). Figure 16b depicts the power consumption graph of the entire drone, with an average of 130 W. In Figure 10, this 130 W is only with 30% of the flying load. The power consumption goes as high as 250 W in higher loads (58% flying load) with simple movements. In maneuvering (Figure 10d–f), the contribution of computation power consumption reduces significantly.

5.2 Offloading SLAM to Hardware Accelerators

Besides preventing lags in the responses of the autopilot, offloading SLAM to a hardware accelerator (i) improves the performance of SLAM and (ii) helps extend the flight time by consuming less power. This section explores these two aspects by implementing SLAM on our three hardware platforms.

SLAM performance. Figure 17 shows the time to process each EuRoC dataset while executing ORB SLAM on a RPi4 (with no other application), TX2, and FPGA. Our FPGA implementation extensively accelerates the local and global bundle adjustments of ORB SLAM (≈90% of execution time on RPi) by using simple modules of dense fixed-size matrix algebra in a pipeline. For further acceleration, we also integrate eSLAM design [50], which accelerates feature extraction. Running SLAM on a separate RPi improves its

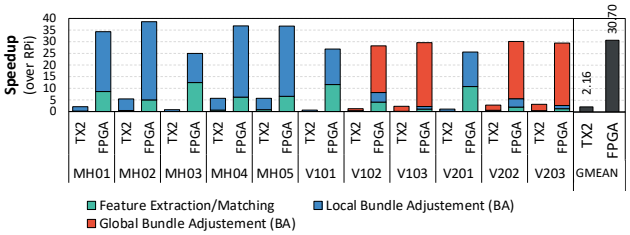


Figure 17: ORB SLAM speedup over RPi for TX2 and FPGA by category: feature extraction and bundle adjustments.

performance by 2.3× (IPC from Figure 15). As Figure 17 illustrates, the TX2 and FPGA implementations are 2.16× and 30.7× faster than the implementation of SLAM on the RPi. As a result, all these implementations, including the slowest, meet the rate of sensors (*e.g.*, cameras and LiDARs), even those with more than 100 FPS. Although all design choices satisfy the real-time requirement, they provide a 400x landscape in power consumption. Therefore, the question is, how should we navigate 400x landscape in power consumption?

Flight time. Offloading SLAM (or any other heavy compute) to a hardware accelerator reduces power consumption, but adds weight to the system. This section explores the combined effect on flight time. The power consumption of our FPGA implantation is 417 mW, compared to ASIC with 24 mW [19]; RPi with 5 W; and TX2 with 10 W. Since, on average, our drone consumes 130 W, saving 10 W by moving from TX2 to FPGA gives us +1 minute of flight time ($\approx$ $^{10}/_{140}$ × 15 min). For small drones moving from CPU/GPU to FPGA with 20x in power savings, there is a reduction on the power consumption of approximately 15-20%, enabling an additional +2–3 minutes of flight time ($\approx$ $^{10}/_{50}$ × 15 min). But, the lengthy process of special ASIC chip fabrication to gain an additional 20x power savings (saving 400 mW) earns us only a few seconds. Table 5 combines the results for the cost of various platforms for executing SLAM on drones by assuming RPi as the baseline. Since TX2 consumes more power and is heavier, the gain in flight time is negative. Both FPGA and ASIC have almost identical impacts on flight time for large drones and small drones (only 20 seconds in additional flight time for ASIC in small drones). However, ASIC integration and fabrication costs are extremely high, which renders FPGA as the best platform even though it consumes more power.

Table 5: Comparing costs of various platforms for SLAM.

Platform		RPi	TX2	FPGA	ASIC
SLAM Speedup		1x	2.16x	30.70x	23.53x
Power Overhead (W)		2	10	0.417	0.024
Weight Overhead (g)		≈50	≈85	≈75	≈20
Integration Cost		Low	Low	Medium	High
Fabrication Cost		Low	Low	Medium	High
Gained Flight Time (min) †	**Small Drones**	0	≈-4	≈2–3	≈2.2–3.2
	Large Drones	0	≈-1.5	≈1	≈1

† Baseline flight time is 15 minutes.

6 RELATED WORK

Prior studies [29, 30] have proposed a closed-loop simulator and benchmark suite for autonomous tasks in drones, mainly focusing on outer-loop tasks, which is not the main focus of this paper. The discussions only pertain to *high-speed* drones. In contrast to the assumptions made, we argue that, first, the mission planning computation does not increase hovering time since mission planning has relaxed deadlines [79]. Even in high-speed, indoor, and cluttered environments, new algorithms have been proposed to enable fast planning [21, 28]. Second, collision detection does not necessarily require heavy computations (*e.g.*, using laser-range, infrared, or RGBD sensors, or even microcontroller) [80–83]. Third, localization is a highly active research area and does not necessarily limit current drone speeds (*e.g.*, real-time odometry and NASA JPL's autonomous racing) [14, 24, 84]. Finally, described conclusions in [29, 30] are based on maximum drone acceleration, the value of which is not readily known from the specifications. Authors have early versions of this work published [85, 86].

7 CONCLUSIONS

This is the first paper that (i) formalized fundamental drone subsystems and quantified how computation power consumption varies in drones and affects the design-space parameters such as flight time; (ii) studied required computing for inner-loop control; and (iii) proposed an open-source drone framework and explored the acceleration landscape of SLAM, while motivating further research within the community. We found that although the outer-loop control is not directly related to real-time control systems due to the nature of heavy computation, it has to consider deadlines; and thus improving the performance of processors is important. For the inner-loop which controls real-time hardware, the amount of computation is relatively low, so low-end embedded computing platforms are satisfactory. However, due to the critical nature of the inner-loop control, all drones have dedicated processors for it. We found that for small drones, improving power efficiency is translated into an increase in flight time, but for heavy drones (>≈2 kg), the improvement in power efficiency does not have an effect. Therefore, FPGA implementations provide the most cost-effective solution for small and large drones.

It is worth mentioning that the studied tradeoffs are different for nano and pico drones with a total power consumption of 100 mW [19, 87–90]. We did not focus on such drones because these drones are extremely customized (from physics to material sciences), so it was not possible to study them within the same framework. Furthermore, we used the minimum TWR of 2. A detailed evaluation for other TWR values can be done in a similar way, released in our repository, which results in a lower contribution of computation power consumption.

ACKNOWLEDGMENTS

We thank the ASPLOS program, artifact-evaluation committees, a cTuning Foundation for their valuable feedback in improving our paper and artifact. Preparing the artifact would not be possible without the extra help from Sam Jijina and Adriana Amyette. We gratefully acknowledge the support of NSF CNS 1815047.

A ARTIFACT APPENDIX

A.1 Abstract

This artifact describes our open-source experimental drone framework that is customizable across its hardware-software stack. The main portion of the artifact focuses on building the drone, which compliments the beginning sections of the paper. The build guide consists of two parts: hardware and software. The hardware guide presents a list of required hardware components (accessible to anyone) following by a step-by-step assembly guide. The software component provides the firmware of the drone and enables users to execute any software that is supported on Linux. We provide the necessary packages and configuration of the software setup. Finally, as an example, we provide simple scripts for `perf` metrics measurements while describing energy consumption measurements (requires an oscilloscope with high-frequency data logging and 30A current probes).

Note: For some artifacts, we provide two links: (1) The original link of the software by the provider; and (2) our copied version path in the open-source repository at the time of preparing this artifact. Please make sure your PDF reader renders hyperlinks.

Note: You can use doi.org/10.5281/zenodo.4546174 to access the most recent version, if any, after publish date.

A.2 Artifact Check-List

- **Algorithm: Drone Firmware**
- **Program: Scripts in python and C++.**
- **Compilation: Python ≥ 2.7 and GCC version 6.3.0.**
- **Transformations:**
- **Binary: Will be compiled on the target platform.**
- **Run-time environment:**
- **Hardware: Raspberry Pi Model 3B+, Emlid Navio2**
- **Execution: command line, bash shell**
- **Metrics: Energy and available `perf` metrics**
- **Output: User measures energy consumption, `perf` produces performance selected performance metric on target platform.**
- **Experiments: Drone energy consumption and autopilot and SLAM `perf` metrics.**
- **How much disk space required (approximately)?: 16 GB SD card**
- **How much time is needed to prepare workflow (approximately)?: Around three hours for building the drone.**
- **How much time is needed to complete experiments (approximately)?: Less than an hour.**
- **Publicly available?: The guide is publicly available with CC BY 4.0 license.**

A.3 Description

A.3.1 How to Access – Hardware. Hardware components can be acquired from any store. The complete list is provided in the hardware dependencies section.

A.3.2 How to Access – Software. The Emlid operating system (OS) image can be accessed here or `/EmlidOS`. For DroneKit, we recommend installing Python pip utility and then obtaining DroneKit from here or `/DroneKit`. MissionPlanner is available for Windows and can be accessed using this link or `/MissionPlanner`.

A.3.3 Hardware Dependencies. The following hardware is required to build the drone and run the experiments/code:

- Raspberry Pi Model 3B+
- Emlid Navio2 Kit
- 4 x 30 Amps ESC
- 4 x MT2213-935KV motors
- RC Controller with receiver
- 4-axis 450mm Drone Frame
- 4 x 1045 Drone Propellers
- 3000mAh 3S LiPo battery
- 915MHz telemetry kit
- PPM encoder
- GPS receiver mount
- 16 GB MicroSD Card
- USB Power Analyzer
- High frequency data analyzing oscilloscope with 30A capable probes

A.3.4 Software Dependencies. The drone OS and software packages are defined below.

- Emlid OS
- Python DroneKit (C++ version of DroneKit can also be utilized)
- MissionPlanner
- Microsoft Windows (Dependency for MissionPlanner; if needed)

A.4 Installation

A.4.1 Drone Assembly. The first steps are to assemble the drone. An overview of the instructions are given below. For a more detailed build guide with pictures please see `/BuildGuide`.

- First assemble the PI + NAVIO. Plug in the HAT into the GPIO pins on the RPI.
- Solder the bullet connectors onto the motor connections.
- Solder the battery connector onto the Power Distribution Board (PDB).
- Screw in the legs of the frame.
- Screw in the top plate to the frame.
- Attach motors to the frame according to the rotational direction listed in the motor manual.
- Use double sided tape and attach Raspberry pi + NAVIO to drone top plate.
- Use double sided tape and attach the PPM encoder to the frame.
- Connect the battery connectors to the PPM encoder.
- Stick the RC receiver onto the frame.
- Connect receiver to the NAVIO.
- Connect PPM outputs to NAVIO.
- Use zip ties and attach ESCs to the bottom of the legs.
- Assemble the GPS mount and zip tie it into the back-right leg (Note : GPS unit must point North-South).
- Attach GPS on mount and connect GPS to NAVIO.
- Connect ESCs to motors and ESCs pwm to PPM encoder.
- Connect battery to battery connector.
- Finally connect TELEM module to HAT and stick module onto frame.

A.4.2 Drone Software Configuration. After building the drone, the following software steps are needed to download and configure the software stack on the drone.

- Download the Emlid OS from here or `/EmlidOS`.
- Flash the downloaded .iso file to the MicroSD card (You can use Etcher as a tool) and insert it into the Pi.
- Follow the first time setup community guide of Arducopter here or `/ArducopterWiki` under "First Time Setup."
- Next, it is critical to configure and calibrate the sensors and IMU. Please follow guide, or `/ArducopterWiki` under "Mandatory Hardware Configuration."
- Expand the filesystem `$sudo raspi-config –expand-rootfs`.

- Install DroneKit `$pip install dronekit`.
- Configure autopilot to load on boot : `$sudo emlidtool –on_boot=True`.
- Review DroneKit docs or `/DroneKitDocs` to see how to use API.
- To spool up Arducopter, run `$sudo systemctl daemon-reload` and then run `$sudo systemctl restart arducopter`.
- Note: RCIO Worker is a background helper service for Arducopter and automatically starts when Arducopter is started.

A.4.3 Setting up and Configuring SLAM.

- Begin by installing Docker `$curl -sSL https://get.docker.com | sh`.
- Add the correct permissions `$sudo usermod -aG docker pi`.
- Install Docker Compose `$sudo pip3 -v install docker-compose`.
- Clone our Github repository or `/ParallelML-Drone`, and change directory to slam `$cd drone/slam`
- Download a sample image data set (here) or `/EuroC-MH01Easy`.
- Extract the data set in the slam directory.
- Run the command `$docker-compose up -d`.
- SLAM is now running in the background.
- To stop SLAM run `$docker-compose down`.

A.5 Experiment Workflow

With a fully working drone, this section describes and provides simple scripts for measuring performance metrics (any performance metric that is available to `perf` tool).

This repository or `/ParallelML-Drone` contain all the required files (and a full backup of our SD card). Specifically, shell scripts `perf_ardu_slam.sh` and `perf_ardupilot_loop.sh` execute simple experiments for Ardupilot and SLAM, respectively. Directory `boot_pi_backup/` contains a backup of our SD card. To use this version, copy the files to SD card and rename it to `boot`.

A.6 Evaluation and Expected Results

Performance Metric Measurements: Execute above scripts by passing the PIDs of `ArduCoptert`, `RCIO_Worker`, and `SLAM` (in this order). Then, the scripts print several metrics for branches, cache operations, and virtual memory management. The exact flags depend on the particular architecture and we have fine-tuned them for Raspberry Pi 3B+.

Energy Measurements: To perform energy measurements an oscilloscope with high-frequency data logging and 30A current probes is required. The current probes are used to measure the current on the input power wires from the LiPo battery. To measure energy (or energy/second), another probe measures the voltage of the battery. By setting the oscilloscope function to multiply these measurements, we can log energy per second of the entire drone. To distinguish between Raspberry Pi, additionally, an in-loop USB power meter to measure Raspberry Pi power consumption is needed. Non-flight measurements can be done while the drone is not active. For flight-related measurements, flip the propellers so the drone pushes down (while consuming a similar amount of energy).

Paper Graphs: You can find the raw data from which the graphs are constructed at `/Drone-CSVs`.

A.7 Experiment Customization

Users are free to change any part of firmware or write their own application for the drone. Additionally, users may add any new sensors or hardware components that is compatible with Raspberry Pi or its GPIO protocols (e.g., I2C).

A.8 Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html

REFERENCES

[1] Ferran Giones and Alexander Brem. From toys to tools: The co-evolution of technological and entrepreneurial developments in drone industry. *Business Horizons*, 2017.

[2] Jong-Hyuk Kim and Salah Sukkarieh. Airborne simultaneous localisation and map building. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 1, pages 406–411. IEEE, 2003.

[3] Matthias Faessler, Flavio Fontana, Christian Forster, Elias Mueggler, Matia Pizzoli, and Davide Scaramuzza. Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle. *Journal of Field Robotics*, 33(4):431–450, 2016.

[4] Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, and Stan Birchfield. Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4241–4247. IEEE, 2017.

[5] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2015.

[6] Shantanu Chaudhary, Arka Prava, Neha Nidhi, and Vijay Nath. Design of all-terrain rover quadcopter for military engineering services. In *Nanoelectronics, Circuits and Communication Systems*, pages 507–513. Springer, 2019.

[7] Nathan Michael, Shaojie Shen, Kartik Mohta, Vijay Kumar, Keiji Nagatani, Yoshito Okada, Seiga Kiribayashi, Kazuki Otake, Kazuya Yoshida, Kazunori Ohno, et al. Collaborative mapping of an earthquake damaged building via ground and aerial robots. In *Field and service robotics*, pages 33–47. Springer, 2014.

[8] Allison Ryan and J Karl Hedrick. A mode-switching path planner for uav-assisted search and rescue. In *Proceedings of the 44th IEEE conference on decision and control*, pages 1471–1476. IEEE, 2005.

[9] H Anil, KS Nikhil, V Chaitra, and BS Guru Sharan. Revolutionizing farming using swarm robotics. In *2015 6th International Conference on Intelligent Systems, Modelling and Simulation*, pages 141–147. IEEE, 2015.

[10] Lian Pin Koh and Serge A Wich. Dawn of drone ecology: low-cost autonomous aerial vehicles for conservation. *Tropical Conservation Science*, 2012.

[11] Si Jung Kim and et al. A survey of drone use for entertainment and avr. In *Augmented Reality and Virtual Reality*, pages 339–352. Springer, 2018.

[12] Shuo Li, Michaël MOI Ozo, Christophe De Wagter, and Guido CHE de Croon. Autonomous drone race: A computationally efficient vision-based navigation and control strategy. *arXiv preprint arXiv:1809.05958*, 2018.

[13] Sunggoo Jung, Sungwook Cho, Dasol Lee, Hanseob Lee, and David Hyunchul Shim. A direct visual servoing-based framework for the 2016 iros autonomous drone racing challenge. *Journal of Field Robotics*, 35(1):146–166, 2018.

[14] NASA JPL. Drone race: Human versus artificial intelligence. nasa.gov/jpl/drone-race-human-vs-ai.

[15] Haomiao Huang, Gabriel M Hoffmann, Steven L Waslander, and Claire J Tomlin. Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering. In *2009 IEEE international conference on robotics and automation*, pages 3277–3282. IEEE, 2009.

[16] Minh-Duc Hua, Tarek Hamel, Pascal Morin, and Claude Samson. Introduction to feedback control of underactuated vtolvehicles: A review of basic control design ideas and principles. *IEEE Control systems magazine*, 33(1):61–75, 2013.

[17] Moses Bangura et al. Aerodynamics and control of quadrotors. 2017.

[18] Simon Lynen, Torsten Sattler, Michael Bosse, Joel A Hesch, Marc Pollefeys, and Roland Siegwart. Get out of my lab: Large-scale, real-time visual-inertial localization. In *Robotics: Science and Systems*, volume 1, 2015.

[19] Amr Suleiman, Zhengdong Zhang, Luca Carlone, Sertac Karaman, and Vivienne Sze. Navion: A 2-mw fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones. *IEEE Journal of Solid-State Circuits*, 54(4):1106–1119, 2019.

[20] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive uav control in cluttered natural environments. In *2013 IEEE international conference on robotics and automation*, pages 1765–1772. IEEE, 2013.

[21] Antonio Loquercio, Ana I Maqueda, Carlos R Del-Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 3(2):1088–1095, 2018.

[22] Ewoud JJ Smeur, Guido CHE de Croon, and Qiping Chu. Gust disturbance alleviation with incremental nonlinear dynamic inversion. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5626–5631. IEEE, 2016.

[23] S Salazar-Cruz, A Palomino, and R Lozano. Trajectory tracking for a four rotor mini-aircraft. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 2505–2510. IEEE, 2005.

[24] Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization. 2017.

[25] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34(3):189–206, 2013.

[26] Shaojie Shen, Yash Mulgaonkar, Nathan Michael, and Vijay Kumar. Multi-sensor fusion for robust autonomous flight in indoor and outdoor environments with a rotorcraft mav. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4974–4981. IEEE, 2014.

[27] William Koch, Renato Mancuso, and Azer Bestavros. Neuroflight: Next generation flight control firmware. *arXiv preprint arXiv:1901.06553*, 2019.

[28] Sikang Liu, Michael Watterson, Sarah Tang, and Vijay Kumar. High speed navigation for quadrotors with limited onboard sensing. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1484–1491. IEEE, 2016.

[29] Behzad Boroujerdian, Hasan Genc, Srivatsan Krishnan, Wenzhi Cui, Aleksandra Faust, and Vijay Reddi. Mavbench: Micro aerial vehicle benchmarking. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 894–907. IEEE, 2018.

[30] Srivatsan Krishnan, Zishen Wan, Kshitij Bhardwaj, Paul Whatmough, Aleksandra Faust, Gu-Yeon Wei, David Brooks, and Vijay Janapa Reddi. The sky is not the limit: A visual performance model for cyber-physical co-design in autonomous machines. *IEEE Computer Architecture Letters*, 19(1):38–42, 2020.

[31] Wikipedia. Mavlink. wiki/MAVLink.

[32] Nathan M Zimmerman. Flight control and hardware design of multi-rotor systems. 2016.

[33] SkyDio. Skydio. skydio.com.

[34] Pixhawk. Pixhawk. pixhawk.org.

[35] Sebastian Scherer, Joern Rehder, Supreeth Achar, Hugh Cover, Andrew Chambers, Stephen Nuske, and Sanjiv Singh. River mapping from a flying robot: state estimation, river detection, and obstacle mapping. *Autonomous Robots*, 33(1-2):189–214, 2012.

[36] DJI. Manifold high-performance embedded computer. dji.com/manifold.

[37] Intel. Intel aero compute board. intel.com/aero-compute-board-guide.

[38] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[39] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16, 2017.

[40] Jakob Engel, Jurgen Sturm, and Daniel Cremers. Semi-dense visual odometry for a monocular camera. In *ICCV*, pages 1449–1456, 2013.

[41] MWM Gamini Dissanayake, Paul Newman, Steve Clark, Hugh F Durrant-Whyte, and Michael Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on robotics and automation*, 17(3):229–241, 2001.

[42] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Dense visual slam for rgb-d cameras. In *IROS*, pages 2100–2106. IEEE, 2013.

[43] Huixiang Chen, Yuting Dai, Rui Xue, Kan Zhong, and Tao Li. Towards efficient microarchitecture design of simultaneous localization and mapping in augmented reality era. In *ICCD*, pages 397–404. IEEE, 2018.

[44] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *ECCV*, pages 834–849. Springer, 2014.

[45] Daniel Törtei Tertei, Jonathan Piat, and Michel Devy. Fpga design and implementation of a matrix multiplier based accelerator for 3d ekf slam. In *ReConFig*, pages 1–6. IEEE, 2014.

[46] Daniel Tertei, Jonathan Piat, and Michel Devy. Fpga design of ekf block accelerator for 3d visual slam. *Computers & Electrical Engineering*, 55:123–137, 2016.

[47] Muhammad Shehzad Hanif, Muhammad Bilal, Khalid Munawar, and Abdullah Saeed Balamash. Implementation of an embedded testbed for indoor slam. In *AICCSA*, pages 1–8. IEEE, 2018.

[48] Leandro de Souza Rosa, Aravind Dasu, Pedro C Diniz, and Vanderlei Bonato. A faddeev systolic array for ekf-slam and its arithmetic data representation impact on fpga. *Journal of Signal Processing Systems*, 90(3):357–369, 2018.

[49] Weikang Fang, Yanjun Zhang, Bo Yu, and Shaoshan Liu. Fpga-based orb feature extraction for real-time visual slam. In *ICFPT*, pages 275–278. IEEE, 2017.

[50] Runze Liu, Jianlei Yang, Yiran Chen, and Weisheng Zhao. eslam: An energy-efficient accelerator for real-time orb-slam on fpga platform. In *DAC*, page 193. ACM, 2019.

[51] Parrot. Parrot anafi. parrot.com/us/drones/anafi.

[52] DJI. Dji mavic. dji.com/mavic.

[53] DJI. Dji spark. dji.com/spark.

[54] DJI. Dji matrice 600. dji.com/matrice.

[55] Parrot. Parrot bebop 2. parrot.com/us/drones/parrot-bebop-2.

[56] iFlight. Succex-e f4 flight controller. iflight-rc.com/SucceX-E-F4.

[57] DJI. Naza-m lite. dji.com/naza-m-lite.

[58] DJI. Naza-m v2. dji.com/naza-m-v2.

[59] Pixhawk. Pixhawk 4. docs.px4.io.

[60] mateksys. Flight controller f405-std. mateksys.com/f405.

[61] Emlid. Emlid navio2 hat for raspberry pi. emlid.com/navio.

[62] Raspberry PI Foundation. Raspberry pi 4. raspberrypi.org/raspberry-pi-4-model-b, 2017. [Online; accessed 04/10/20].

[63] NVIDIA. Nvidia jetson tx. nvidia.com/jetson-tx2, 2017. [Online; accessed 04/10/20].

[64] RunCam. Runcam night eagle 2 pro. runcam.com/uncam-night-eagle-2-pro/.

[65] emesent. Hovermap. emesent.io/hovermap/.

[66] YellowScan. Yellowscan surveyor. yellowscan-lidar.com/lidar-solutions/.

[67] Velodyne Lidar. Ultra puck. velodynelidar.com/ultra-puck/.

[68] Parrot. Parrot mambo mission. parrot.com/us/drones/mambo.

[69] DJI. Dji mavic air. dji.com/mavic-air.

[70] ArduPilot. Ardupilot. ardupilot.org.

[71] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[72] DroneKit. Dronekit api. dronekit.io.

[73] Thomas Nagy. Waf. waf.io.

[74] Michael Kerrisk. Linux daemon. linux/man-pages/daemon, 2019. [Online; accessed 04/10/20].

[75] Bitcraze. Crazyflie 2.1. bitcraze.io/crazyflie-2-1, 2019. [Online; accessed 04/10/20].

[76] Drona Aviation. dronaaviation.com/plutox.

[77] Connor Soohoo and Yunchih Chen. Orb-slam2 gpu optimization. github.com/connorsoohoo/ORB-SLAM2-GPU-RGBD, 2017. [Online; accessed 04/10/20].

[78] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[79] Philemon Sakamoto. *UAV mission planning under uncertainty*. PhD thesis, Massachusetts Institute of Technology, 2006.

[80] Adam Bry, Abraham Bachrach, and Nicholas Roy. State estimation for aggressive flight in gps-denied environments using onboard sensing. In *2012 IEEE International Conference on Robotics and Automation*, pages 1–8. IEEE, 2012.

[81] Nils Gageik, Paul Benz, and Sergio Montenegro. Obstacle detection and collision avoidance for a uav with complementary low-cost sensors. *IEEE Access*, 3:599–609, 2015.

[82] Kimberly McGuire, Guido De Croon, Christophe De Wagter, Karl Tuyls, and Hilbert Kappen. Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone. *IEEE Robotics and Automation Letters*, 2(2):1070–1076, 2017.

[83] Bardienus P Duisterhof, Srivatsan Krishnan, Jonathan J Cruz, Colby R Banbury, William Fu, Aleksandra Faust, Guido CHE de Croon, and Vijay Janapa Reddi. Learning to seek: Autonomous source seeking with deep reinforcement learning onboard a nano drone microcontroller. *arXiv preprint arXiv:1909.11236*, 2019.

[84] Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. Showcasing real-time visual-inertial odometry. youtube.com/UZH-Robotics.

[85] Sam Jijina, Adriana Amyette, Nima Shoghi, Ramyad Hadidi, and Hyesoon Kim. Understanding the software and hardware stacks of a general-purpose cognitive drone. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 212–214. IEEE, 2020.

[86] Sam Jijina, Adriana Amyette, Ramyad Hadidi, and Hyesoon Kim. Towards a general purpose cognitive drone. In *The Fourth Workshop on Cognitive Architectures (CogArch 2020)*.

[87] Robert J Wood, Ben Finio, Michael Karpelson, Kevin Ma, Néstor Osvaldo Pérez-Arancibia, Pratheev S Sreetharan, Hiro Tanaka, and John Peter Whitney. Progress on 'pico'air vehicles. *The International Journal of Robotics Research*, 31(11):1292–1302, 2012.

[88] Kevin Y Ma, Pakpong Chirarattananon, Sawyer B Fuller, and Robert J Wood. Controlled flight of a biologically inspired, insect-scale robot. *Science*, 340(6132):603–607, 2013.

[89] Dario Floreano and Robert J Wood. Science, technology and the future of small autonomous drones. *Nature*, 521(7553):460–466, 2015.

[90] Avinash Singh, Thomas Libby, and Sawyer B Fuller. Rapid inertial reorientation of an aerial insect-sized robot using a piezo-actuated tail. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4154–4160. IEEE, 2019.