Roofline Model for UAVs: A Visual Performance Model for Guiding Compute System Design in Autonomous Drones

“All models are wrong, but some are useful.” – George Box.

Srivatsan Krishnan†, Zishen Wan†, Kshitij Bharadwaj†, Aleksandra Faust§, and Vijay Janapa Reddi†

†Harvard University
‡LLNL
§Google Brain Research

Abstract—We present a bottleneck analysis tool for designing compute systems for autonomous Unmanned Aerial Vehicles (UAV). The tool provides insights by exploiting the fundamental relationships between various components in the autonomous UAV such as sensor, compute, body dynamics. To guarantee safe operation while maximizing the performance (e.g., velocity) of the UAV, the compute, sensor, and other mechanical properties must be carefully designed (or selected). The goal of our proposed tool is to provide a visual model which aids system architects to understand optimal compute design (or selection) for autonomous UAVs. The tool is available here: https://bit.ly/skyline-tool

I. INTRODUCTION

Autonomous machines like unmanned aerial vehicles (UAVs) are on the rise [1]–[6]. Amongst these, quadcopters account for the vast majority of the market [7] due to their ability to take-off and land vertically and navigate in confined spaces. These unique capabilities enable them to be deployed in numerous applications, such as search and rescue [3], [8], package delivery [9], [10], surveillance [11], and sports photography [12]. These unique applications have motivated the industry to build domain-specific hardware [13]–[16].

However, building computing systems that can run the autonomy algorithms for UAVs efficiently is challenging because autonomous UAVs differ from traditional computing systems (embedded systems, servers, etc.). They are severely size, weight, and power (SWaP) constrained. Moreover, UAVs have many components like sensors, compute mechanical frames, and rotors interacting with one another to work as one coherent system. The selection of these components greatly affects the UAV system’s “performance” (i.e., velocity, mission time, energy). For instance, components such as the sensor framerate and associated data processing rate determine how fast the UAV reacts in a dynamic environment, determining its safe velocity. Likewise, payload weight and thrust, which is the property of mechanical components like frame and rotors, determine if the physics allows it to accelerate and move faster.

Given that UAVs are complex autonomous systems, and that the computing platform is just one component among many others, to design the optimal onboard compute we need to consider the role of compute in the context of the whole UAV system. As we demonstrate, traditional raw compute performance metrics can lead to misguided conclusions. Therefore, to guide systematic design, we need tools that help us understand the interactions between these components and quantify them.

In this paper, we introduce a bottleneck analysis tool to guide the design of optimal computing systems for UAVs. The tool determines which of the UAV components (compute, sensor, physics) determines the safe operating velocity; safe high-speed autonomous navigation remains one of the key challenges in enabling aerial robot applications [17]–[20]. A high safe velocity ensures that the UAV is reactive to a dynamic environment and ensures that the UAV finishes tasks quickly, thereby lowering mission time and energy [21].

Figure 1 shows that the tool takes the compute algorithm and hardware as input along with the UAV type to perform bottleneck analysis. The coupling between the different choices leads to different “rooflines (i.e. bounds).” For example, the AscTec Pelican [24] (red) has a different roofline from DJI Spark [25] (blue) where the $y$-axis is the maximum safe velocity and the $x$-axis is the action throughput (i.e., how fast the UAV can make decisions). The individual design points on the chart reflect the maximum safe velocity for a given UAV system (i.e., autonomy algorithm, compute hardware, and UAV type) configuration. The output visually resembles that of a traditional computer system roofline model [26]; however, unlike the roofline model, the parameters in our model quantify...
the UAV as a holistic system as opposed to compute system in isolation. Similar to the roofline model, our tool can be used by computer architects in two ways. First, it can be used as a visual performance model to understand various bounds and bottlenecks in the UAV system. Second, it can guide architects towards building an optimal compute system (autonomy algorithm + on-board compute).

We use the tool to answer a wide array of characterization questions that are relevant for architects designing compute systems for autonomous UAVs. How does the ad-hoc/off-the-shelf selection of onboard compute affect a UAV’s safe velocity? Given a fixed UAV configuration and onboard compute payload, how do we systematically evaluate different autonomy algorithms to maximize the UAV’s performance? Given a UAV type and onboard payload, how do we evaluate between different sensors and how they affect the UAV’s performance? Given a UAV type, autonomy algorithms, sensor choices, how do we systematically characterize redundancy in onboard compute so as to maximize the UAV’s safe velocity? Finally, given several onboard compute platforms, autonomy algorithms, and sensors choices, how do we characterize and select components that maximize the UAV’s safe velocity and how does this selection differ as we change the UAV types?

All our characterization studies reveal the important intertwined relationship between various components in UAVs and mission performance. The current state-of-the-art techniques rely on selecting (or designing) onboard compute in an ad-hoc fashion or based on metrics such as peak compute throughput (or higher FPS), low-power without considering its true impact on UAV performance. For instance, our model shows that selecting or designing onboard compute in this fashion can result in 1.4× to 2× degradation in safe velocity. Hence, a systematic methodology is needed to characterize these complex UAV systems. To that end, our work introduces a modeling tool that gives fundamental insights into how component selections impact the UAV’s safe velocity. Moreover, they also offer optimization targets for computer architects designing onboard compute for workload targeting autonomous UAVs.

In summary, we make the following contributions:

1) A roofline-like performance model called F-1 for UAV system characterization.
2) An interactive tool for the F-1 model called Skyline provides various insights into different bounds and bottlenecks as UAV components (parameters) change.
3) Using the Skyline tool, we perform a detailed characterization study by varying each UAV component and its effect on overall performance.
4) A complete system-level characterization of UAVs with commercial onboard compute choices and hardware accelerators explicitly built for autonomous UAVs. Our study reveals that ad-hoc design choices solely based on compute throughput, low power, or energy efficiency can be misleading and shows architects how much optimization is needed to maximize the UAVs’ performance.

II. AUTONOMOUS UAVS

Our work primarily targets quadcopters; henceforth, we will refer to these systems as UAVs for brevity. This section provides a background on the critical components in a UAV system.

A. UAV Components

Autonomous UAVs typically have three key components (Figure 2a), namely rotors, sensors, and an onboard computing platform. Rotors determine the thrust a UAV can generate. The sensor allows the UAV to sense the environment. A common type of sensor used in UAV is a camera. The computer executes the autonomy algorithm based on sensor data. The physical size of a UAV plays an important role in the selection of the components. Also, the size of the UAV imposes constraints on the maximum weight of each component. For instance, a bigger UAV will have the capability to have multiple sensors and powerful onboard computing platform. In contrast, a smaller UAV will have limited sensor and computing capabilities.

B. Size, Weight, and Power (SwaP) Constraints

The flight time and energy available to carry out missions can vary drastically based on the UAV’s size, as shown in Figure 2b. The battery capacity and its endurance (flight time) are commensurate with the size of the UAV. For instance, a mini-UAV (e.g., Asctec Pelican) frame size is typically in the order of 350 mm and higher and has a battery with a power rating excess of 3830 mAh. In contrast, nano-UAVs (e.g., Bitcraze CrazyFlie) have the frame size, around 7mm or less, and have a battery with 240 mAh or less. As the battery capacity decreases, so do its endurance.

C. Onboard Compute

The SwaP constraints have implications on the onboard compute. Table I lists a sample of commonly used UAV platforms by roboticists and the onboard compute used in those UAVs. On one extreme we have the nano-UAVs that,

---

**Fig. 2:** (A)-(B) Machine components, implications of size on battery and flight time. (C) The sensor-compute-control pipeline which determines the decision making rate, based on bottleneck analysis model [22], [23].
due to their size and weight, typically use microcontrollers as the onboard computing platform. For example, Bitcraze CrazyFlie [27] weighs less than 27 g and is powered by an ARM Cortex-M4 microcontroller. On the other end, mini-UAVs, which are bigger and have a higher weight (payload capacity). A mini-UAV typically uses a general-purpose computing platform. For example, AscTec Pelican [24], which weighs 1.6 Kgs and is powered by a general-purpose Intel NUC platform.

D. Flight Controller

While the primary onboard compute is responsible for making high-level decisions (path planning and trajectory generation etc.), the low-level task of stable flight and control is delegated to a dedicated flight controller which are realized using PID controllers. The flight controller firmware stack is computationally light and is typically run on the microcontrollers [28], [29] such as Arm Cortex M4 [28], [29]. The flight controller uses the onboard sensors, such as the Inertial Measurement Unit (IMU) [30] and GPS, to stabilize and control the UAV. To stabilize the UAV from unpredictable errors (sudden winds or damaged rotors), the inner-loop typically runs at closed-loop frequencies of up to 1 kHz [31], [32].

E. Autonomy Algorithms

The autonomy algorithms process the raw sensor data on the onboard compute to determine the high-level actions that are later translated to the flight controller. These algorithms can be classified into two main categories, namely “Sense-Plan-Act” (SPA) and “End-to-End Learning” (E2E). In SPA, the algorithm is broken into three or more distinct stages, namely the “sensing stage”, the “planning stage”, and the “control stage”. In the sensing stage, the sensor data is used to create a map [38]–[41] of the environment. The planning stage [42], [43] then uses the map, to determine the best trajectory (e.g., collision-free, lowest energy, etc.). The trajectory information is used by the control stage, which actuates the rotor. In E2E, the methods process the raw input sensor information (such as RGB, Lidar, etc.) and use a neural network model to directly produce output actions. Unlike the SPA paradigm, the end-to-end learning methods do not require maps or separate planning stages. The model can be trained using supervised learning [35], [44]–[46] or reinforcement learning [34], [47]–[49].

III. AN INSIGHTFUL VISUAL PERFORMANCE MODEL

In this section, we introduce the F-1 visual performance model that helps computer architects understand whether a UAV’s performance is bottlenecked by compute (and autonomy algorithm), or by other components in the UAV such as sensor or its physics. We first start with the F-1 model overview to understand it as a performance model, and explain how it can be useful. We then describe how we construct the F-1 model.

A. A Visual Performance Model

The F-1 model establishes a roofline-like relationship between the safe velocity of the UAV and its decision-making rate. To intuitively understand safe velocity, let us consider a simple scenario depicted in Figure 3a where a UAV has a sensor with a sensing distance of ‘d’ meters and a field of view (FoV) [50] denoted by the green region. Any obstacle (e.g., tree or a bird) within the FoV can be sensed by the UAV, and the autonomy algorithm running on an onboard compute platform can make appropriate decisions at the rate of \( f_{\text{action}} \). Considering all the payload (sensor, compute, battery, etc.), the UAV’s physics can allow it to accelerate by up to \( a_{\max} \). In such cases, the UAV can safely fly up to a velocity of \( V_{\text{safe}} \) such that even if an obstacle is within its FoV, it can safely stop without colliding. Any velocity greater than \( V_{\text{safe}} \) means the UAV cannot stop safely and will collide with the obstacle. Achieving high safe velocity is important as it lowers the mission time and overall mission energy [51].

The decision-making rate (‘Action Throughput’) determines how fast the UAV can sense, process, and take appropriate actions. It can be calculated as the throughput of the sensor-compute-control pipeline shown in Figure 3b. As the stages in the sensor-compute-control pipeline can be run concurrently, the minimum latency of the pipeline can never be smaller than the maximum latency of each component in the subsystem:

\[
\max(T_{\text{sensor}}, T_{\text{compute}}, T_{\text{control}}) < T_{\text{action}} \quad (1)
\]

If the stages of the pipeline are not fully overlapped, the total pipeline latency can never exceed:

\[
T_{\text{action}} \leq T_{\text{sensor}} + T_{\text{compute}} + T_{\text{control}} \quad (2)
\]

Thus, between Eq 1 and Eq 2 we have the upper bound and lower bound in the pipeline latency. From these relationship we can estimate the upper-bound on the Action Throughput \( f_{\text{action}} \) using the bottleneck analysis [22], [23].
compute, we can estimate the compute throughput \( f_{\text{compute}} \) equal to the sensor’s frame rate \( f_{\text{s}} \)

Limit the safe velocity \( V_s \) to prevent the UA from exceeding its physical limitations. The choice of onboard sensors may also affect the decision making rate (\( f_{\text{sensor}} \)) and the knee-point throughput \( f_k \). The sensor-bound case occurs when the compute throughput \( f_{\text{compute}} \) is greater than the sensor throughput \( f_{\text{s}} \) (i.e., action throughput is equal to \( f_{\text{s}} \) according to Eq. 3), and \( f_{\text{s}} < f_{\text{k}} \). In this scenario, the sensor adds a new ceiling to the F-1 model, thus, bounding the velocity under \( V_s \). In this region, the compute throughput is improved (e.g., higher FPS sensor), the velocity cannot exceed the sensor-bound ceiling \( V_s \) no matter how fast onboard compute can process the sensor input.

**Compute Bound.** The choice of onboard compute (or autonomy algorithm) also affects the decision making rate \( f_{\text{action}} \). Figure 4a shows that a UAVs velocity is compute-bound if its compute throughput \( f_{\text{compute}} \) is less than the sensor’s frame rate \( f_{\text{s}} \) and the knee-point throughput \( f_k \). The computing platform adds a new ceiling to the roofline model, bounding the velocity under this limit \( V_c \). Unless the compute throughput is improved (e.g., hardware accelerators/algorithm-hardware co-design) the velocity cannot exceed \( V_c \).

**Physics Bound.** A UAV’s physical properties such as weight, thrust produced by its rotors determine how fast it can move. Hence, the ultimate bound on the safe velocity \( V_{\text{safe}} \) will be determined by its physics, or what is sometimes referred to as body dynamics. We call the region to the right of the knee-point (i.e., when sense-to-act throughput is greater than or equal to \( f_k \)) as Physics bound. Unless the physical components are improved (e.g., increasing thrust-to-weight ratio), the velocity cannot exceed the critical safe peak speed velocity no matter how fast a decision is made (i.e., faster compute/sensor).

**Sensor Bound.** The choice of onboard sensors may also limit the decision-making rate \( f_{\text{action}} \) which in turn can limit the safe velocity \( V_{\text{safe}} \). As shown in Figure 4a, a robot’s velocity is sensor-bound if its action throughput is equal to the sensor’s frame rate \( f_{\text{s}} \) but less than the knee-point throughput \( f_k \).

The choice of onboard sensors may also affect the decision making rate \( f_{\text{sensor}} \) and the knee-point throughput \( f_k \). The sensor-bound case occurs when the compute throughput \( f_{\text{compute}} \) is greater than the sensor throughput \( f_{\text{s}} \) (i.e., action throughput is equal to \( f_{\text{s}} \) according to Eq. 3), and \( f_{\text{s}} < f_{\text{k}} \). In this scenario, the sensor adds a new ceiling to the F-1 model, thus, bounding the velocity under \( V_s \). In this region, the compute throughput is improved (e.g., higher FPS sensor), the velocity cannot exceed the sensor-bound ceiling \( V_s \) no matter how fast onboard compute can process the sensor input.

**Compute Bound.** The choice of onboard compute (or autonomy algorithm) also affects the decision making rate \( f_{\text{action}} \). Figure 4a shows that a UAVs velocity is compute-bound if its compute throughput \( f_{\text{compute}} \) is less than the sensor’s frame rate \( f_{\text{s}} \) and the knee-point throughput \( f_k \). The computing platform adds a new ceiling to the roofline model, bounding the velocity under this limit \( V_c \). Unless the compute throughput is improved (e.g., hardware accelerators/algorithm-hardware co-design) the velocity cannot exceed \( V_c \).

**Physics Bound.** A UAV’s physical properties such as weight, thrust produced by its rotors determine how fast it can move. Hence, the ultimate bound on the safe velocity \( V_{\text{safe}} \) will be determined by its physics, or what is sometimes referred to as body dynamics. We call the region to the right of the knee-point (i.e., when sense-to-act throughput is greater than or equal to \( f_k \)) as Physics bound. Unless the physical components are improved (e.g., increasing thrust-to-weight ratio), the velocity cannot exceed the critical safe peak speed velocity no matter how fast a decision is made (i.e., faster compute/sensor).

**Sensor Bound.** The choice of onboard sensors may also limit the decision-making rate \( f_{\text{action}} \) which in turn can limit the safe velocity \( V_{\text{safe}} \). As shown in Figure 4a, a robot’s velocity is sensor-bound if its action throughput is equal to the sensor’s frame rate \( f_{\text{s}} \) but less than the knee-point throughput \( f_k \).

The relationship between the safe velocity \( V_{\text{s}} \) plotted on the y-axis and Action Throughput \( f_{\text{action}} \) plotted on the x-axis results in a roofline-like visual performance model as shown in Figure 4a. The model can be expanded to provide meaningful abstractions such as bounds and bottleneck analysis for computer architects designing onboard compute.

**B. Sensor, Compute and Physics Bounds**

The model can be used to perform a bound-and-bottleneck analysis to determine if the safe velocity is affected by the onboard sensor/compute or the UAV’s physics. Any point to the left of the “knee-point” in F-1 (Figure 4a) denotes that the safe velocity is bounded by the choice of compute (and autonomy algorithms) or sensor, and any point to the right of the knee-point denotes the velocity is bounded by physics of the UAV. Ideally, to achieve the optimal pipeline design, it’s action throughput should be equal to that of the knee-point.

**Physics Bound.** A UAV’s physical properties such as weight, thrust produced by its rotors determine how fast it can move. Hence, the ultimate bound on the safe velocity \( V_{\text{safe}} \) will be determined by its physics, or what is sometimes referred to as body dynamics. We call the region to the right of the knee-point (i.e., when sense-to-act throughput is greater than or equal to \( f_k \)) as Physics bound. Unless the physical components are improved (e.g., increasing thrust-to-weight ratio), the velocity cannot exceed the critical safe peak speed velocity no matter how fast a decision is made (i.e., faster compute/sensor).

**Sensor Bound.** The choice of onboard sensors may also limit the decision-making rate \( f_{\text{action}} \) which in turn can limit the safe velocity \( V_{\text{safe}} \). As shown in Figure 4a, a robot’s velocity is sensor-bound if its action throughput is equal to the sensor’s frame rate \( f_{\text{s}} \) but less than the knee-point throughput \( f_k \). The sensor-bound case occurs when the compute throughput \( f_{\text{compute}} \) is greater than the sensor throughput \( f_{\text{s}} \) (i.e., action throughput is equal to \( f_{\text{s}} \) according to Eq. 3), and \( f_{\text{s}} < f_{\text{k}} \). In this scenario, the sensor adds a new ceiling to the F-1 model, thus, bounding the velocity under \( V_s \). In this region, the compute throughput is improved (e.g., higher FPS sensor), the velocity cannot exceed the sensor-bound ceiling \( V_s \) no matter how fast onboard compute can process the sensor input.

**C. F-1 Model for Quantifying Optimal Compute Designs**

Every UAV configuration has a unique F-1 model, thus resulting in a unique knee-point. Recall that the knee-point is the minimum action throughput required to maximize the safe velocity. Therefore, we can use this information to determine an ideal or optimal onboard compute (or autonomy algorithm) for a given UAV system. Furthermore, if the compute system design is sub-optimal, it helps us understand the performance gap between the current compute design and the optimal design.

**Optimal Design.** For a given UAV with fixed mechanical properties, changing the sensor type or onboard compute (autonomy algorithm) affects the \( f_{\text{action}} \). Consequently, the optimal design point is when the action throughput that is equal to the knee-point throughput \( f_k \) as shown in Figure 4b.

**Over-Optimal Design.** If the action throughput is \( f_{\text{over}} \) such that \( f_{\text{over}} > f_k \), then either the sensor/computer is over-optimized since any value greater than \( f_k \) yields no improvement in the velocity of the UAV. Such an over-designed computing/sensor platform involves not only extra optimization effort but can also burn additional power that further increases the UAV’s total power, decreasing its overall battery life.

**Sub-Optimal Design.** If the action throughput is \( f_{\text{sub}} \), such that \( f_{\text{sub}} < f_k \), then the sensor/computer is under-optimized, which signifies that the system if off by \( f_{\text{sub}} - f_k \) and there is scope for improvement through a better algorithm or selection (or design) of the computing system as shown in Figure 4b.
D. Constructing the F-1 Visual Performance Model

In this section, we describe how we construct the F-1 model starting from prior work [24] that has established and validated the relationship between the UAV component parameters and the safe velocity of a UAV as described by the equation below:

\[ v_{\text{safe}} = a_{\text{max}} \left( \sqrt{T_{\text{action}}^2 + 2d} / a_{\text{max}} - T_{\text{action}} \right) \]  

(4)

Eq. 4 states that if the UAVs physics can permit it to accelerate at most by \( a_{\text{max}} \), its compute and sensors permit it to sense and act at an interval of \( T_{\text{action}} \) (\( 1/f_{\text{action}} \)), and its sensor(s) can sense the environment as far as ‘d’ meters, then robot can travel as fast as \( V_{\text{safe}} \).

To construct the model, we sweep the \( T_{\text{action}} \) from 0 → 5 s along with typical accelerations values (\( a_{\text{max}} = 50 \) m/s\(^2\)) and the sensor range (\( d = 10 \) m), as shown in Figure 5a. We observe an asymptotic relation between velocity and \( T_{\text{action}} \) such that as \( T_{\text{action}} \rightarrow 0 \), the velocity \( \rightarrow 32 \) (as seen in the magnified portion of Figure 5b). Likewise, as the \( T_{\text{action}} \rightarrow \infty \), the velocity \( \rightarrow 0 \). To derive F-1 from Figure 5a, we plot the \( f_{\text{action}} \) (inverse of \( T_{\text{action}} \)) on the \( x\)-axis and velocity on the \( y\)-axis in Figure 5b. There is a point beyond which increasing \( f_{\text{action}} \) does not increase the velocity, showing a roofline.

We annotate the plots with two sample points denoted as point ‘A’ and ‘knee-point’. The point A has a \( f_{\text{action}} \) of 1 Hz while the knee-point has a \( f_{\text{action}} \) of 100 Hz. Between point A to knee-point denotes 100× improvement in action throughput and translates to increase in velocity from 10 m/s to 30 m/s. Whereas even 100× improvement in \( f_{\text{action}} \) after the knee-point results in 1.0004× improvement in velocity (signifying no improvement in velocity). Hence, increasing the action throughput (e.g., faster computing platform, faster sensor etc.) beyond the knee-point yields no improvement in \( V_{\text{safe}} \).

E. Relationship between Safe Velocity and Number of Missions

An important operational efficiency metric for autonomous drones is the ‘number of missions’, which captures how many times the drone can complete similar missions on a single battery charge. For example, in a package delivery use case, a higher number of missions means more packages delivered with lower downtime spent recharging. This metric is affected by the choice of drone onboard compute combined with several other key components.

For a given drone, we define the number of missions as:

\[ N_{\text{missions}} = \frac{E_{\text{battery}}}{E_{\text{mission}}} \]  

(5)

where \( E_{\text{battery}} \) is the total energy available in the drone (a function of battery mAh rating) and \( E_{\text{mission}} \) is the total energy expended by the drone per mission.

We can define \( E_{\text{mission}} \) for a single mission as:

\[ E_{\text{mission}} = (P_{\text{rotors}} + P_{\text{compute}} + P_{\text{others}}) \times t_{\text{mission}} \]  

(6)

where \( P_{\text{rotors}}, P_{\text{compute}}, \) and \( P_{\text{others}} \) refer to the power consumption of the rotor propulsion, compute, and other electronic components (e.g., sensors, ESC) in the drone.

F. Experimental Validation, Other Models and Modeling Error

The underlying relationship outlined in Eq 4, which forms the basis of F-1, is inherently a safety model for the UAV. It is based on the premise that a given UAV can sense the obstacle and stop before colliding [24], provided its physics allows it. Since the parameters in a UAV such as sensing distance, decision-making rate, and acceleration/deceleration (depends upon UAVs thrust, payload, and physics) can all be pre-determined, we can also calculate the upper-bound on the UAV velocity such that it can safely stop. The premise behind Eq. 4 is validated across different environments with obstacle densities ranging from 0 to 50, and wind speeds up to 7 m/s on a mini-UAV. Furthermore, the model is validated in both simulations as well as in a real-world setting. \(^1\)

There are also other safety models [52] where they assume that the UAV does not have a stopping policy and therefore

\(^1\)https://www.youtube.com/watch?v=or0e2HFLACEY4
cannot reduce its forward velocity (longitudinal velocity). These models also account for UAV parameters such as sensing distance, acceleration capability, sensing latency, and size of the obstacles. Interestingly, these safety models also result in the same trend between velocity and decision-making rate results as shown in Figure 5a. These models have also been validated on an actual UAV in lab environments. However, determining the size of the obstacle is non-trivial in a real-world setting. Hence, we select the safety model [24] described in Eq. 4 which does not make any assumptions about the obstacle size.

Since the F-1 model is derived by plotting the safety model [24] between velocity ($V_{safe}$) and throughput ($f_{action}$), we build upon a thoroughly validated model. The only source of error stems from approximating the curve (i.e., the region where safe velocity increases with action throughput) with a straight line which introduces an error in the range of 1-3% depending upon the UAV configuration.

It is important to note that these prior safety models [24], [52] for UAVs do not provide bottleneck analysis nor provide necessary abstractions that can guide computer architects on the design of onboard compute. Likewise, in the computer architecture community, onboard compute designed for UAVs [33], [35], [53] does not consider the effects of UAV parameters but instead, quantify based on compute throughput or low-power. However, we later show that (Section V) designing onboard compute based on these metrics alone can impact sub-optimal UAV performance. Hence F-1 is an essential abstraction built upon a safety model for UAVs which can help architects building onboard compute for UAVs.

### G. Effects of UAV Component Interaction

A UAV is a complex system comprising sensors, compute, electromechanical components such as rotors, batteries, etc. The effect of these components that affect decision making rate can be abstracted by the $T_{action}$ ($1/f_{action}$) in Eq. 4. The components that affect the UAV physics, such as the mass of sensor/compute/body frame/battery, the thrust-to-weight ratio, the aerodynamic effects such as drag [54], sensing quality, etc. can be abstracted by the $a_{max}$ and $d$ parameters in Eq 4. Generally, the three parameters ($T_{action}$, $a_{max}$, $d$) in Eq. 4, can be used to capture overheads of improving safety, reliability, and redundancy. In Section V, we show how F-1 model can be used for modeling these scenarios.

The $a_{max}$ parameter captures the physical effects of adding payload (sensor, onboard compute, battery, etc.) to the UAV. The payload weight affects the thrust-to-weight [55] ratio which lowers the $a_{max}$ [56]. The F-1 model captures the impact of varying $a_{max}$ on $V_{safe}$: a higher $a_{max}$ leads to a higher $V_{safe}$ (with roofline shifting upwards), as shown in Figure 4c.

The $d$ parameter captures the sensing quality of the UAV. For instance, a laser based sensor can provide a higher sensing range, whereas a camera array based depth sensor has a limited range [57]. The F-1 model captures the impact of varying $d$ on $V_{safe}$: a higher $d$ leads to higher $V_{safe}$ (with roofline and slope shifting upwards), as shown in Figure 4d.

| Parameter         | SI Unit | Description         |
|-------------------|---------|---------------------|
| Sensor Framerate  | Hz      | Throughput of the sensor. |
| Compute TDP       | W       | Maximum TDP of the onboard compute. Used to design the heatsink. |
| Autonomy Algorithm| N/A     | Select a pre-configured autonomy algorithm. |
| Compute Runtime   | s       | Measures the latency of the autonomy algorithm. Used to calculate compute throughput. |
| Sensor Range      | m       | Maximum range of the sensor. |
| Drone Weight      | g       | Maximum weight of the UAV without any extra payload. |
| Rotor Pull        | N       | Measures the thrust produced by the rotor propulsion. |
| Payload Weight    | g       | Total weight of the payload including onboard compute, sensors, battery etc. |

TABLE II: Summary of all the knobs (cyber and physical parameters) available in the Skyline tool.

Lastly, the $f_{action}$ parameter captures the effect of sensor framerate, improvements to autonomy algorithm, or onboard compute. The additional latency incurred due to extra sensor/computation (e.g., sensor-fusion) affects the $f_{action}$ based on Eq. 3. The F-1 model captures the impact of varying $f_{action}$ by adding new ceilings which will limit the $V_{safe}$.

In summary, Eq. 4 couples interactions of all UAV components and their associated effects into a single relationship. Thus F-1 model, which is built based on Eq. 4 provides a unified performance model for computer architects to design onboard compute with a holistic view of the UAV system.

### IV. SKYLINE: AN INTERACTIVE VISUAL TOOL FOR F-1

We provide a web-based tool (Skyline) for the F-1 model to make it easy to use. The tool gives the end-user the ability to do exploratory (what-if) studies on the impact of changing various parameters of the UAVs such as autonomy algorithms or changing the onboard compute and sensor.

#### A. Overview

The tool has three major components. The first component includes the interactive knobs (sliders) for various UAV components and their parameters. The second component is the visualization area that dynamically plots the F-1 model (Section III) ties the various UAV component interactions together to determine the safe velocity of the UAV. The maximum safe velocity has implications on the overall mission performance [21] (e.g., mission time, energy, etc.). Table II summarizes the list of available knobs in the Skyline tool.

The tool also provides user-defined knobs for sensor range, sensor framerate, compute platform TDP (which is used to

https://bit.ly/skyline-tool
TABLE III: Overview of the evaluation in Section V. The table outlines the goals of each study and what parameters are varied or fixed. The highlighted parameter cell is varied while the rest of the UAV’s parameters are kept constant.

C. Visualization Area

The visualization area plots the F-1 model based on the UAV configurations. The interactive UAV parameter knobs result in a dynamic plot that gives the user the fundamental understanding of how these parameters affect the UAV’s safe velocity. For example, increasing the TDP of the compute platform increases the heatsink weight, increasing the payload weight. As payload weight increases, the maximum acceleration decreases, which lowers the safe velocity. Likewise, changing the sensor framerate affects the decision-making rate, determining how fast or slow the UAV can travel. Thus, using the dynamic visualization plot of the F-1 model enables the user to intuitively understand the effect of changing UAV parameters.

D. Analysis

The Skyline tool has an analysis section where it outputs information such as knee-point throughput, safe velocity achievable for a given UAV. In addition to the performance data, the analysis section also provides information about the fundamental limits in safe velocity (i.e., sensor bound, compute-bound, or physics bound) and several optimization tips.

V. EVALUATION

We present several case studies (Table III) on how to use the Skyline tool for the characterization of various components in a UAV. The analysis of these characterizations is used to gather architectural insights that can build efficient onboard compute platforms. From Section V-A to Section V-D, we first show how to use the Skyline tool for characterizing individual components in the UAV system while keeping the rest of the parameters fixed. Then, in Section V-E, we demonstrate how the tool is used for complete UAV system characterization.

![A Different onboard compute.](image1)

![F-1 Plot.](image2)

FIG. 6: Case study of choosing between Intel NCS and Nvidia AGX for DJI Spark running DroNet.

A. Onboard Compute Characterization

In this case study, we answer the following research questions: How does adhoc selection of onboard compute affects the UAVs performance? Onboard compute is a vital component for achieving autonomy in UAVs. However, despite its importance, the selection (and even design, as we show in Section VI) of onboard compute remains ad-hoc [24], [33], [58]. To that end, we characterize two ad-hoc selections of onboard compute and see its impact on UAVs performance.

**UAV Configurations.** We select the pre-configured ‘DroNet’ [35] autonomy algorithm in our tool. We then select the DJI Spark pre-configured UAV. For onboard compute, we toggle between Intel NCS and Nvidia AGX to see the F-1 plots in the visualization area. The Skyline tool internally calculates the heatsink weight [59] based on the TDP of these systems. We keep the sensor FPS at 60 Hz to ensure that we are not in the sensor-bound region for both these UAVs.

**Analysis.** The annotated version of the two plots is combined into a single plot and show in Figure 6b. Based on the F-1 plots for the onboard compute platform, for the DJI spark, the Nvidia AGX has a lower roofline than the DJI spark with Intel NCS. The higher roofline with the Intel NCS is because the onboard compute weighs less than the Nvidia AGX, which means the UAV’s physics restricts it from achieving higher safe velocity than the DJI spark with the Intel NCS. However, even though Nvidia AGX can achieve 1.5× more compute throughput than Intel NCS running DroNet, the high compute performance of Nvidia AGX cannot translate to a higher safe velocity as the physics of the UAV does not allow it.

To improve the respective UAV’s performance, we can lower the power (or TDP) of Xavier AGX at the cost of performance (since AGX is over-provisioned by 33×). This will ensure that the TDP is reduced, resulting in a smaller heat-sink weight (PCB components). Or one can also translate the performance gains of AGX to faster safe-velocity by increasing the thrust to weight ratio (i.e., either lower the payload weight or increase the thrust by using a motor with greater pull).

**Takeaway.** Selection of onboard cannot be ad-hoc or chosen based on compute performance in isolation, and one must consider the effects of all the UAV parameters and understand what implications the compute has on these parameters. A high-performance compute does not necessarily translate into a high performing UAV. Skyline can help architects understand the fundamental role of computing in autonomous UAVs.
B. Autonomy Algorithm Characterization

In this case study, we answer the following research questions: Given a fixed UAV configuration and onboard compute payload, what is the effect of different autonomy algorithms on $V_{safe}$? The primary function of an autonomy algorithm is to make intelligent decisions to achieve the mission goals while also guaranteeing safety (e.g., avoid colliding with an obstacle). However, despite its importance, the selection or design of the autonomy algorithm is often made in isolation without considering the impact of onboard compute or UAV components. So we characterize how changing autonomy algorithms impact UAVs performance.

**UAV Configurations.** We consider two autonomy algorithm paradigms, namely SPA (Section II) and E2E, and evaluate it on an AscTec Pelican UAV with TX2. We keep the UAV, onboard compute fixed, and evaluate different autonomy algorithms to understand their impact on the UAVs safe velocity.

**Analysis.** The resulting three plots are combined into a single F-1 plot and shown in Figure 7b. The SPA achieves a compute throughput of 1.1 Hz on Nvidia TX2. However, due to its low decision-making rate, the maximum safe velocity achievable by the UAV is limited to 2.3 m/s. On the other hand, in the case of TrailNet and DroNet (E2E algorithms), they achieve a compute throughput of 55 Hz and 178 Hz on Nvidia TX2 which achieves a higher safe velocity.

For an AscTec Pelican UAV with the TX2 onboard compute, the knee-point throughput is 43 Hz, suggesting that both TrailNet and DroNet are over-provisioned by $1.27 \times$ and $4.13 \times$. In this case, the high compute throughput of the autonomy algorithm does not translate to a higher safe velocity unless the UAV’s physics changes. However, for the SPA paradigm, the UAV’s safe velocity is compute-bound. Therefore, the compute throughput needs to be improved by $39 \times$ to achieve the optimal safe-velocity permitted by the UAV’s physics.

**Takeaway.** When designing autonomy algorithms for UAVs (assuming the onboard compute is fixed), commonly used metrics like higher compute throughput or energy efficiency can be misleading and can result in over-optimization (which can increase the design cost) or under-optimization (affects robot performance). In over-optimized scenario the higher compute performance might not result in higher UAV performance. In under-optimized case, the Skyline tool can give computer architects the performance targets to achieve as part of their optimization efforts, thereby helping us design optimal systems.

C. Modular Redundancy Characterization

In this case study, we answer the following research questions: Given a UAV platform with fixed autonomy algorithms and sensor choices, how do we systematically characterize redundancy in onboard compute and its impact on UAV’s performance? UAVs need to operate robustly in highly dynamic environments [60], and redundancy in compute or sensor ensures safety in the event of a failure (or corruption). Sometimes dual redundancy [61] (or triple redundancy [62]) increases reliability where a majority vote determines the final decision. Though redundancy can increase reliability, it can also increase the cost and negatively affects the UAV’s performance. So we characterize the effects of adding redundancy in onboard compute and evaluate its effects on the UAV’s safe velocity.

**UAV Configurations.** In this case study, we keep the UAV, type, sensor payload fixed and evaluate how adding dual compute redundancy affects the UAV’s safe velocity. Using the Skyline tool, we first select UAV [46] from the pre-configured autonomy algorithms. From the pre-configured UAVs, we select AscTec Pelican. For sensors, we will assume an RGB-D camera with a frame rate of 60 FPS and a sensing distance of 4.5 m. In this UAV configuration, we evaluate the effects of having single Nvidia-TX2 and dual Nvidia-TX2 SoC sharing the same board and evaluate the effects of dual compute redundancy. For estimating the baseline safe velocity, we select the pre-configured Nvidia-TX2 from the pre-configured platform. To evaluate the effects of having dual redundancy in onboard compute (as shown in Figure 8a), we assume that another Nvidia-TX2 is added to the UAV platform and has the sensor input. The output from these two computing platforms is validated and then sent to the controller (similar to Tesla’s FSD stack). For modeling this scenario using the Skyline tool, we account for the payload weight for the additional TX2, which includes the computing platform and the heatsink weight. By selecting the Custom (User-Defined) setting under UAV body settings, we change the “Sensor and other component weight” to account for the additional TX2 platform.

**Analysis.** The annotated version of the resulting two plots is combined into a single plot and shown in Figure 8b. Since the same autonomy algorithm (and onboard compute) is used in both these UAVs, it achieves a compute throughput of 178 Hz running on Nvidia TX2 (annotated as the red dotted line in Figure 8b). The baseline scenario (single onboard compute) is annotated as “Roofline-TX2” in the Figure 8b. For the dual compute redundancy, the increase in payload weight (heatsink
board) lowers the maximum acceleration capability of the UAV, which lowers the roofline (annotated as “Roofline-2×TX2” in Figure 8b), thereby reducing the safe velocity by 33% compared to the baseline. Thus, there is a trade-off between enabling redundancy and robot operational performance.

To bridge the performance gap, architects can replace the over-provisioned with an onboard computer with $\frac{1}{2}$th of compute throughput (for DroNet), thereby lowering the TDP, which will help accommodate two onboard compute within same power envelope reducing the payload (heatsink) weight.

**Takeaway.** Redundancies improve autonomous UAVs’ safety with the downside of increasing deployment costs and lower UAV performance (e.g., safe velocity). To fully understand the consequences of redundancies, there is a need to systematically characterize the effects of additional payload and its impact on a UAV’s decision-making rate and physics. Skyline tool can give architects the intuition into various bottlenecks and performance targets to build safe yet efficient redundancy systems.

### D. Sensor Characterization

In this case study, we answer the following research question: *Given a UAV with fixed onboard compute, how do we systematically evaluate different sensors and their impact on UAV’s performance?* How far the sensor can detect obstacles determines how fast the UAV can travel and stop safely before colliding with an obstacle. Sensors like RGB-D captures the image along with the depth information. The commercially available RGB-D cameras can achieve 90 FPS with a sensing distance of 4.5 m. However, sensors like RGB cameras don’t have native depth information and rely on software solutions to determine the depth. We characterize the effect of different sensors, namely, RGB (depth estimation in software) and RGB-D camera, and its impact on UAV’s safe velocity.

**UAV Configurations.** We consider the AscTec Pelican UAV with Intel NCS onboard compute. In this UAV configuration, we evaluate two different sensors namely RGB-D (depth) and RGB camera. We first select the Nvidia TX2 from the pre-configured platforms. We then select TrailNet [46] from the pre-configured autonomy algorithms. To evaluate these sensors, we consider scenarios shown in Figure 9a. RGB-D camera natively produces the depth image. For example, Primesense camera can produce depth images with a range of up to 4.5 m [24] at 90 FPS. In RGB camera, the depth information is not available and software techniques for depth estimation is commonly used. A neural network based depth estimation can predict depths up to 10 m to 40 m [63] depending upon the environment. The increased sensing distance also incurs additional latency to process the neural network depth estimation kernel.

We can model the trade-off between these sensor choices using the Skyline tool. Running the depth estimation model [63] on TX2 has a latency of 91 ms, which lowers the sensor framerate to 11 FPS. We take a conservative depth estimation of 10 m. Using the custom (user-defined) setting in the Skyline tool, we slide the sensor range knob and sensor FPS knobs to 4.5 m and 60 Hz, respectively, for the RGB-D camera. We slide the sensor range knob and sensor FPS knobs to 10 m and 11 Hz for the RGB camera with depth estimation.

**Analysis.** The annotated version of the resulting two plots is combined and shown in Figure 9a. Since the same autonomy algorithm (and onboard compute) is used in both these UAVs, it achieves a compute throughput of 51 Hz running on Nvidia TX2 (annotated as the red dotted line in Figure 9a). Furthermore, in the case of AscTec Pelican with RGB-D Sensor, the safe velocity is bounded by physics as denoted by the “Roofline-TX2”. Thus, in this case, using a 60 FPS framerate is over-provisioned by almost $2\times$. Thus, in this scenario, the sensor framerate can be safely lowered to 30 FPS without impacting the safe velocity of the UAV. In contrast, for the RGB + Depth estimation, the safe velocity is bounded by the sensor framerate (ceiling due to the sensor is annotated in Figure 9a), suggesting that improving the sensor system throughput by $2\times$ will increase the safe-velocity by 25%. Nonetheless, comparing the RGB-D (with d=4.5 m and 60 FPS) and RGB + Depth (with d=10 m with 11 FPS) head-on, the latter has a higher roofline suggesting it can achieve a higher safe velocity by having a larger sensing distance (Figure 4d).

**Takeaway.** Selecting sensors for UAVs based solely on metrics such as high sensor framerate can be misleading and can result in over-provisioning it (increasing power or cost). Unless the safe velocity is bounded by sensor framerate, any improvements to it will not impact the final UAV’s performance. Hence, it is important to consider other intrinsic parameters for the sensor (such as sensing distance) along with UAV parameters. Our tool gives designers guidance on selecting sensors and how that affects the UAV’s overall performance.

### E. Putting it All Together: Full System Characterization

In this case study, we answer the following research question: *Given several onboard compute platforms, autonomy algorithms, and sensors choices, how do we systematically characterize and select components to maximize the UAV’s safe velocity? How does this selection differ as we change the UAV types?* This is in contrast to what we have seen this far. Thus far, we saw how to characterize different onboard compute...
platforms, sensors, and autonomy algorithms, individually, while keeping other UAV parameters fixed.

**UAV Configurations.** We consider several different choices for each UAV. For onboard compute choices, we consider Nvidia TX2, Nvidia AGX, Ras-Pi, Intel NCS. Similarly, for different UAV types, we consider AscTec Pelican and DJI Spark. Finally for autonomy algorithms, we consider DroNet [35], TrailNet [33], and CAD2RL [34]. Using the Skyline tool, we characterize each combination (i.e., sensor, autonomy algorithm, onboard compute, and UAV type). We also share insights from the tool on how we can optimize each of these combinations.

**Analysis.** The annotated version of all these results are combined into a single plot and shown in Figure 10b. Based on the F-1 plots for various combinations, we can classify these designs as compute-bound designs or physics-bound designs. Below, we discuss what computer architects can do if the design point falls into the compute-bound/physics-bound category.

**Compute-Bound Designs.** In compute-bound designs, the safe velocity is bounded by the compute throughput of the autonomy algorithm (i.e., the UAV cannot make decisions faster). For designs in this region, computer architects can apply the traditional optimization techniques ranging from micro-architectural to algorithm to improve the compute throughput. For instance, in the case of the AscTec Pelican UAV, Ras-Pi does not have sufficient computing capability to run any of the autonomy algorithm (DroNet, TrailNet, and CAD2RL) under consideration. Therefore, any new architectural optimization for Ras-Pi (e.g., building a custom hardware accelerator within Ras-Pi system) would need to improve the compute throughput by 3.3× for DroNet, 110× for TrailNet, and 660× for CAD2RL.

**Physics-Bound Designs.** In physics-bound design, the safe velocity is bounded by the physics of the UAV rather than autonomy algorithm or onboard compute performance. If the designs in this region lower the roofline (e.g., TX2 for DJI Spark), then computer architects can help alleviate this problem by optimizing for lower TDP. For instance, in the case of DJI Spark with TX2, as an onboard compute running DroNet, it achieves a compute throughput of 178 Hz. However, the knee-point for this UAV is 30 Hz suggesting that it is over-provisioned by a factor of 6×. Computer architects can trade off this excess performance for lower TDP (e.g., running the system at a lower clock frequency) so that its heatsink weight can be reduced, lowering the overall payload weight.

**Takeaway.** Ad-hoc selection of UAV components or designing them in isolation impacts UAV’s performance. Hence, a systematic methodology is needed to characterize these UAV systems holistically. To that end, Skyline tool can give fundamental insights into how various component selections impact the UAVs performance. Moreover, they also provide optimization goals for computer architects designing onboard compute for workload targeting autonomous UAVs.

### VI. Pitfalls in Designing Hardware Accelerators

This section discusses the common pitfalls of designing hardware accelerators for UAVs based purely on metrics such as ‘compute throughput’ and ‘low-power.’ We consider two popular hardware accelerators, namely Navion [53] and PULP-DroNet [64] and characterize them for a nano-UAV [65] with Skyline. Based on our characterization, we demonstrate the need for a systematic approach in understanding the performance requirement needed for a particular UAV rather than designing a high-performance compute in isolation.

**Navion.** Navion is a hardware accelerator for visual-inertial odometry (a portion in SLAM without loop-closure) in the SPA paradigm, which achieves a throughput of 172 FPS in 2mW. Using Navion in an autonomous UAV still requires other algorithms like integration of maps (e.g., Octomap), motion planning, and control. Figure 11a, shows how Navion chip will be used in an end-to-end autonomous UAV application. For the other stages in the SPA pipeline, we characterize the package-delivery application in MavBench [21] and replace the SLAM portions with performance numbers reported in Navion paper [53]. This estimate is still optimistic for Navion, considering it doesn’t perform loop-closure.

**PULP-DroNet.** PULP is designed for accelerating DroNet [35]. Unlike Navion, this PULP can enable end-to-end autonomous operations in UAVs since DroNet directly operates on raw sensor information to control the UAV.

**Analysis.** We characterize the Navion and PULP as onboard compute along with other components a the nano-UAV [65]. We also assume the sensor framerate is 60 FPS. The F-1 plot for these two configurations are shown in Figure 11c.

The PULP achieves a throughput of 6 FPS @ 64 mW, which results in a compute-bound scenario. The performance of the PULP system has to be increased by 4.33× to achieve a peak safe-velocity capable by the nano-UAV. In the case of the Navion chip, even though the SLAM stage achieves an impressive performance of 172 FPS @ 2 mW, integrating into the complete SPA pipeline (Figure 11a) increases the overall latency to 810 ms (625 ms for octomap on TX2, 175 ms...
for motion-planning on TX2 and 1 ms for flight controller) or action throughput of 1.23 Hz. Since the UAV knee-point throughput is around 26 Hz, this accelerator also results in a compute-bound scenario. The end-to-end throughput of the SPA pipeline must be improved by $21.1 \times$ to achieve the peak safe-velocity of the UAV. That improvement should target building accelerators for octomap and path-planning.

**Takeaway.** Optimizing compute based on traditional metrics like throughput, low power, or energy efficiency can be misleading, and one needs to consider the compute performance along with the entire autonomy pipeline and other UAV components. To that end, our tool provides architects intuitions and bottleneck analysis that can be used in optimization efforts in computer system design that translate to UAVs capabilities.

**VII. Related Work**

Performance models, such as the Multicore Amdhal’s law [66], Roofline model [26], and Gables [23], are useful to guide the design of an optimal system for a given workload. These models are applicable for traditional compute systems and not for UAV systems. Our work proposes a roofline-like model to help understand the role of computing in UAVs.

For performance modelling of complex systems (i.e., beyond compute only system), cote [67] is a full-system model for design and control of nano-satellites. The cote model takes into account orbital mechanics, understanding physical bounds on communication, computation, and data storage to design a cost-effective, low-latency, and scalable nano-satellite system. The F-1 model has a similar objective, where it combines the interactions between compute/sensor, and physics to understand various bottlenecks to build an optimal UAV system.

**VIII. Conclusion**

Autonomous vehicles are on the rise. Building computing systems for this emerging domain requires us to understand the role of computing in the context of the rest of the physical system. We need new tools and methods to engineer computing systems that are balanced (not over-designed, nor under-designed) with respect to the overall cyber-physical system capabilities. This paper focuses on quadrotors and shows that it is essential to quantify the role of computing in designing efficient compute subsystems for SWaP-constrained autonomous UAVs. Our tool identifies the bottlenecks (sensor, compute, body dynamics). It gives architects a measure of how much the computing system is over- or under-designed concerning the rest of the UAV system configuration. We expect architects to use the tool to understand how much their domain-specific hardware will improve a given UAV’s safe velocity, as that ultimately impacts mission time, energy and the number of sorties that the UAV can achieve. We also expect architects to use the tool to help them explore the design space and generate the optimal domain-specific processor architecture that is best suited for a UAV design.

**References**

[1] A. Timothy, M. N. Paul, A. T. Aaron, B. Joan, and S. Jeff, “Drone transportation of blood products,” TRANSFUSION Journal, vol. 57, no. 3, pp. 582–588, 2017.

[2] R. Feltman, “The future of sports photography:drones..” https://www.theatlantic.com/technology/archive/2014/02/how-drones-are-helping-nepal-earthquake-relief-effort/.

[3] J. Rogers, “How drones are helping the nepal earthquake relief effort.” http://www.foxnews.com/tech/news/2015/04/30/how-drones-are-helping-nepal-earthquake-relief-effort.html.

[4] M. Bacco, A. Berton, E. Ferro, C. Gennaro, A. Gotta, S. Matteoli, F. Paonessa, M. Ruggeri, G. Virono, and A. Zanella, “Smart farming: Opportunities, challenges and technology enablers,” in 2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany), pp. 1–6, May 2018.

[5] Z. Wan, B. Yu, T. Y. Li, J. Tang, Y. Zhu, Y. Wang, A. Raychowdhury, and S. Liu, “A survey of fpga-based robotic computing.” IEEE Circuits and Systems Magazine, vol. 21, no. 2, pp. 48–74, 2021.

[6] S. Liu, Z. Wan, B. Yu, and Y. Wang, “Robotic computing on fpgas,” Synthesis Lectures on Computer Architecture, vol. 16, no. 1, pp. 1–218, 2021.

[7] https://www.grandviewresearch.com/industry-analysis/global-commercial-drones-market, title = Commercial Drone Market Size, Share & Trends Analysis Report By Application (Filming & Photography, Inspection & Maintenance), By Product (Fixed-wing, Rotary Blade Hybrid), By End Use, And Segment Forecasts, 2019 - 2025, year = 2020.

[8] A. Qiantori, A. B. Sutiono, H. Hariyanto, H. Suwa, and T. Ohta, “An camera visual system for autonomous machines on the early stages of a natural disaster in indonesia,” J. Med. Syst., vol. 36, 2012.

[9] E. Weise, “Amazon delivered its first customer package by drone.” https://www.usatoday.com/story/tech/news/2016/12/14/amazon-delivered-its-first-customer-package-drone/95401366/.

[10] H. Michel, “Amazon’s drone patents.” http://dronecenter.bard.edu/amazon-drone-patents/.

[11] D. R. C. McCullough, “Unmanned aircraft systems(uas) guidebook in development.” https://cops.usdoj.gov/html/dispatch/08-2014/UASGuidebookinDevelopment.asp.

[12] R. Feltman, “The future of sports photography:drones..” https://www.theatlantic.com/technology/archive/2014/02/the-future-of-sports-photography-drones/283896/.

[13] “Deploy ai-powered autonomous machines at scale.”

[14] “Nvidia jetson tx2 module.” https://developer.nvidia.com/embedded/jetson-tx2.

[15] “News & announcements.” https://www.movidi.us/news/intel-movidius-3d-camera-advanced-computer-vision-and-deeplearn, May 2017.

[16] Z. Wan, Y. Zhang, A. Raychowdhury, B. Yu, Y. Zhang, and S. Liu, “An energy-efficient quad-camera visual system for autonomous machines on fpga platform.” In 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS), pp. 1–4, IEEE, 2021.

[17] “Darpa fla.” https://www.darpa.mil/program/fast-lightweight-autonomy.
[18] K. Mohta, K. Sun, S. Liu, M. Watterson, B. Pfrommer, J. Svacha, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, “Experiments in fast, autonomous, gpus-denied quadrotor flight,” in 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 7832–7839, IEEE, 2018.

[19] G. Loianno, D. Scaramuzza, and V. Kumar, “Special issue on high-speed vision-based autonomous navigation of uavs,” Journal of Field Robotics, vol. 1, no. 1, pp. 1–3, 2018.

[20] S. Li, M. M. Oz, C. De Waeger, and G. C. de Croon, “Autonomous drone race: A computationally efficient vision-based navigation and control strategy,” arXiv preprint arXiv:1809.05958, 2018.

[21] B. Boroujerdi, H. Genc, S. Krishnan, W. Cui, M. Almeida, K. Man-soorshahi, A. Faust, and V. J. Reddi, “Mavbench: Micro aerial vehicle benchmarking,” in 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 894–907, 2018.

[22] M. D. Hill, “Three other models of computer system performance,” CoRR, vol. abs/1901.02926, 2019.

[23] M. Hill and V. J. Reddi, “Gables: A roofline model for mobile socs,” in 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 317–330, IEEE, 2019.

[24] S. Liu, M. Watterson, S. Tang, and V. Kumar, “High speed navigation for quadrotors with limited onboard sensing,” in 2016 IEEE International Conference on Robotics and Automation (ICRA), pp. 1484–1491, IEEE, 2016.

[25] “Dji spark.” https://www.dji.com/spark, month = 04, year = 01, note = Accessed on (04/01/2020).

[26] S. Williams, A. Waterman, and D. Patterson, “Roofline: an insightful visual performance model for multicore architectures,” Communications of the ACM, vol. 52, no. 4, pp. 65–76, 2009.

[27] https://www.bitcraze.io/products/crazyflie-2-1/, title = CrazyFlie 2.1 Product Page year = 2020.

[28] “Pixhawk 1 flight controller.” https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk.html, 2020.

[29] “All about multicopter drone fpv flight controllers.” https://www.getfpv.com/learn/new-to-fpv/all-about-multicopter-fpv-drone-flight-controller/, 2020.

[30] M. Achtelik, T. Zhang, K. Kuhlenhun, and M. Buss, “Visual tracking and control of a quadcopter using a stereo camera system and inertial sensors,” in 2009 International Conference on Mechatronics and Automation, pp. 2863–2869, IEEE, 2009.

[31] D. Gurjan, D. Stumpf, M. Achtelik, K.-M. Doth, G. Hirzinger, and D. Rus, “Energy-efficient autonomous four-rotor flying robot controlled at 1 khz,” in Proceedings 2007 IEEE International Conference on Robotics and Automation, pp. 361–366, IEEE, 2007.

[32] W. Koch, R. Mancuso, and A. Bestavros, “Neuroflight: Next generation flight control firmware,” arXiv preprint arXiv:1901.06553, 2019.

[33] G. S. Smolyanskiy, A. Kamenev, J. Smo and S. Birchfield, “Field of view,” https://en.wikipedia.org/wiki/Field

[34] D. Gonzalez, J. Perez, V. Milanes, and F. Nashashibi, “A review of motion planning techniques for automated vehicles,” IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 4, pp. 1135–1145, 2016.

[35] R. Mahony, V. Kumar, and P. Corke, “Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor,” IEEE Robotics and Automation magazine, vol. 19, no. 3, pp. 30–32, 2012.

[36] https://www.gnc.nasa.gov/www/k-12/airplane/wrat.html, title = Thrust to Weight Ratio, year = 2020.

[37] https://www.bitcraze.io/products/crazyflie-2-1/, title = CrazyFlie 2.1 Product Page year = 2020.

[38] “Heat sink size calculator.” https://celsiaiinc.com/resources/calculators/heat-sink-size-calculator/, (Accessed on 01/29/2020).

[39] Y.-S. Hisiao, Z. Wan, T. Jia, R. Ghosal, A. Raychowdhury, D. Brooks, G.-Y. Wei, and V. J. Reddi, “Mavfi: An end-to-end fault analysis framework with anomaly detection and recovery for micro aerial vehicles,” arXiv preprint arXiv:2103.12374, 2021.

[40] P. Bannon, G. Venkataramanan, D. D. Sarma, and E. Talpes, “Computer and redundancy solution for the full self-driving computer,” in 2019 IEEE Hot Chips 31 Symposium (HCS), pp. 1–22, IEEE Computer Society, 2019.

[41] A. Mehmed, M. Antlanger, and W. Steiner, “The monitor as key architecture element for safe self-driving cars,” in 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S), pp. 9–12, IEEE, 2020.

[42] S. A. Y. Fang, H. Abu-Haimed, K. Lien, D. Fu, and J. Gu, “Learning object-specific distance from a monocular image,” CoRR, vol. abs/1909.04182, 2019.

[43] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, “A 64mw dnn-based visual navigation engine for autonomous nano-drones,” IEEE Internet of Things Journal, 2019.
[65] X. Zhang, B. Xian, B. Zhao, and Y. Zhang, “Autonomous flight control of a nano quadrotor helicopter in a gps-denied environment using on-board vision,” IEEE Transactions on Industrial Electronics, vol. 62, no. 10, pp. 6392–6403, 2015.
[66] M. D. Hill and M. R. Marty, “Amdahl’s law in the multicore era,” Computer, vol. 41, no. 7, pp. 33–38, 2008.
[67] B. Denby and B. Lucia, “Orbital edge computing: Nanosatellite constellations as a new class of computer system,” in Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 939–954, 2020.